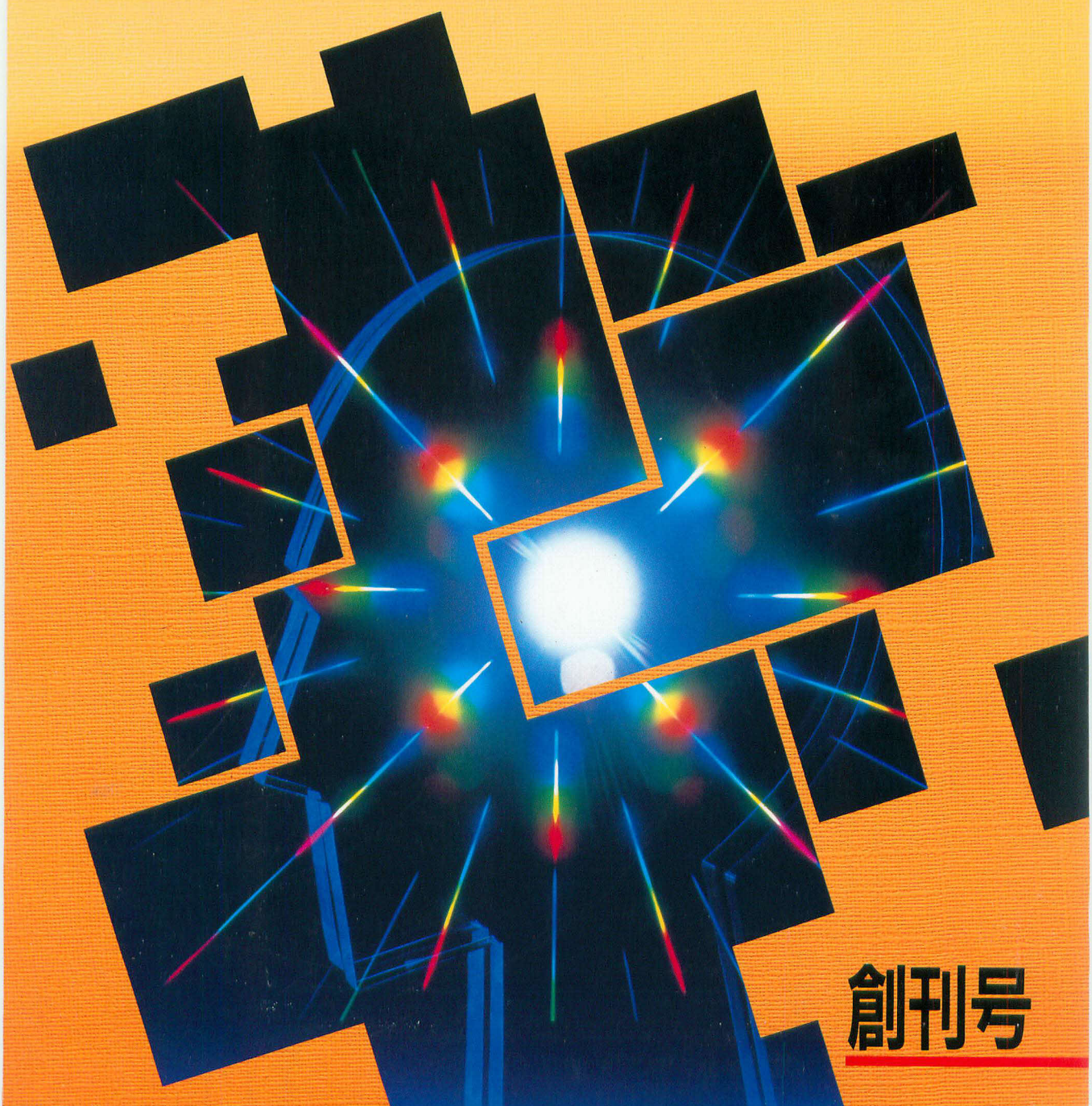
別冊トランジスタ技術

トラ技コンピュータ

特集 MS-DOSとデバイス・ドライバ

- デバイス・ドライバとは
- ブロック型/キャラクタ型デバイス・ドライバの作り方
- 常駐型デバイス・ドライバTSR/EMSドライバ/他

1989 11

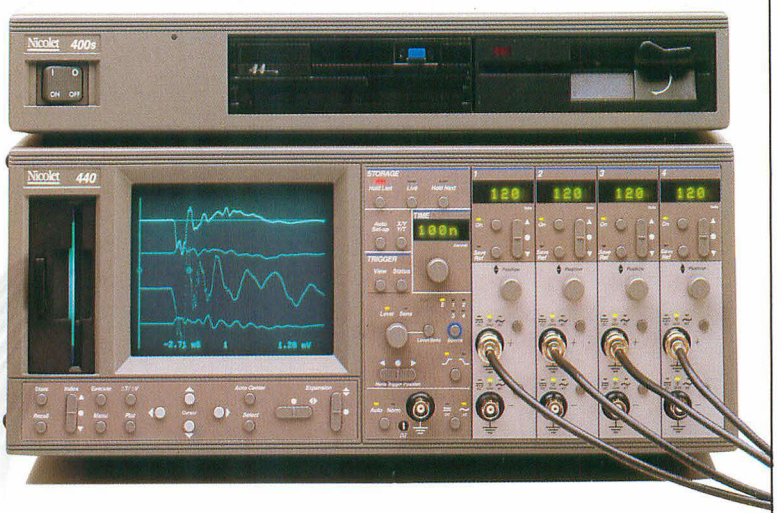


創刊号

次世代のデジタル・オシロスコープ!!

デジタル信号処理を1台で簡単に。

'89 JEMIMA
T&M EXHIBITION
日本電子計測展
11月8日(水)～10日(金)
大阪マーチャンダイズ・マート(OMM)
社 日本電気計測器工業会



単に波形観測(モニタ)を行なう時代から、波形処理する時代の到来です。

NicoletのSystem400シリーズは、デジタルの特長をフルに活かした、ディスク供給プログラムやMS-DOSフォーマットの波形保存機能で、従来のような複雑なプログラミングを必要とせずに、何時でも簡単にユーザが望む形でデータを取り出すことができます。この解析プログラムは定期的に開発され、ソフトウェア・ライブラリとしてフロッピ・ディスクで供給されます。さらに、これらの解析プログラムを接続(リンク機能)することによって、本体の自動制御や演算処理を簡単に行なうことができます。

●本体内蔵プログラム
工学単位変換(最大3文字、リアルタイム変換)

●標準プログラム
加減乗除算、微積分、スムージング、標準偏差値、実行値、平均値、面積値、立ち上がり時間、勾配、絶対値、水平移動、等

●近日発売予定
単純加算平均、FFT(128K)、境界線判定、ピークホールド

〈仕　様〉

モデル	430	440	450	460
チャンネル数	2ch	4ch	2ch	4ch
入力方式	差動入力		シングルエンド	
垂直分解能	12ビット(0.025%)		8ビット(0.4%)	
サンプリング速度	10MS/秒(100nS)		200MS/秒(5ns)	
メモリ容量	64Kワード/ch(256Kワード/chオプション)			
定価(消費税別)	¥2,700,000	¥3,800,000	¥2,200,000	¥3,500,000

※定価は、1989年10月1日 現在

デジタル・オシロスコープ
System 400 シリーズ

Nicolet
INSTRUMENTS OF DISCOVERY

ニコレージャパン株式会社
〒153 東京都目黒区東山1丁目1番2号 東京☎03(715)2551 大阪☎06(863)1550・名古屋☎052(741)2150

資料請求No.2

トラ技コンピュータ CONTENTS 1989年11月号

TREND

◆パソコン用CADソフトは実用になるか ……37
◆光ディスクと対応ソフトの動向 ……41
◆80386の仮想86モードを使うとできること ……45
◆ハイパ・メディアの現況 ……47
◆Xウィンドウがもたらす世界 ……49

特集 MS-DOSとデバイス・ドライバ

簡単に各種の周辺装置を接続することができる理由
第1章 MS-DOSのデバイス・ドライバの役割 ● 井芹陽一 ……54

I/Oポートにつながる RAMディスクを作る
第2章 ブロック型デバイス・ドライバとRAMディスクの作り方 ● 藤井 敦
1 ハードウェアの製作 ……65
2 デバイス・ドライバの構造 ……73
3 RAMディスク・デバイス・ドライバの作成 ……84

時計表示と縮小印字ドライバを作る
第3章 キャラクタ型と常駐型デバイス・ドライバの作り方 ● 石垣武彦 ……92
<コラム> Macintoshのデバイス・ドライバ ……98
PARTNER-S/Hを使ったデバイス・ドライバのデバッグ ……102

EMMマネージャを使うための基本をマスタする
第4章 拡張メモリ用EMSドライバの使い方 ● 菱 博嘉 ……108
<コラム> 機種依存性などに関して ……112

2バイト・コードの処理は案外複雑
第5章 PC9801のIO.SYSの解析 ● 斎藤健司 ……116
<コラム> デバイス・ドライバの常駐を知るツール ……118
OS-9のデバイス・ドライバ ……121
MS-DOS 3.3のPRINT.SYS ……122

漢字とANKの比率を1:2にしたままでバイカ文字以外でも使える
第6章 プリンタ・ピッチを変えるユーティリティの作り方 ● 中野正次 …125

連載

◆MS-DOS上でのマクロ・アセンブラの開発手順 ● 相沢一石 ……131
<コラム> ファイル型式……135 マクロ命令……136

第1回 Cによる実践プログラミング技術入門
◆ANSIのCと標準関数の詳細 ● 大貫広幸 ……144

第1回 システム開発技術者のためのPascalプログラミングの手解き
◆Turbo Pascal入門 ● 中野正次 ……161

第1回 技術者のためのdBASE IIIプログラミング入門
◆メニュ・プログラムの作成 ● 丹羽一夫 ……174

一般

●エンジニアへのブレテン・ボード(君にも作れる高性能計算機アーキテクチャ) ……130
●初心者ならPascalをおすすめします ……159
●コンピューティング・エッセイ(今 フラクタルがおもしろい) ……158

トラ技コンピュータ創刊にあたって

「トランジスタ技術」誌は，その名のとおりトランジスタから始まり，リニアIC，OPアンプ，ゲート論理IC，マイクロプロセッサとまさに集積回路の発展と共に歩んできました．しかし，時代の歩みは速く，ICを基本とするハードウェアに加えて，ソフトウェアが登場し，その比重が急激に高まりつつあります．「トランジスタ技術」誌のみでは，このソフトウェアの分野まで十分にカバーしきれないと予想されます．そこで，「トランジスタ技術」誌と同じ視点でソフトウェアを取り上げるために，「トラ技コンピュータ」を創刊いたしました．

特集テーマは，マイクロプロセッサ利用/応用技術，プログラミング技術，データ処理技術などを中心に取り上げる予定です．

読者と共に成長する雑誌にしたいと考えておりますので，ご愛読のほどよろしくお願いいたします．

特集について

デバイス・ドライバについての説明は，多くの雑誌や書籍で取り上げられています．しかし，系統だてて，どういうふうに作っていけばよいか，何故このような構造になっているのかなどの疑問について，初心者にも理解できるようには説明されていなかったのではないでしょうか．今回の特集を読めば，きっとあなたにもPDSに登録できるぐらいの作品が作れるようになるでしょう．あとは，アイデアしだいです．

投稿募集のお知らせ

ソフトウェアは，ハードウェアに比べると作成者の個性が大きく現れます．よいプログラムを作ることを心がけたいものですが，自分だけではなかなかわからないものです．そこで，役に立つプログラムを作ったり，よいアイディアが浮かんだら，ぜひ，お知らせください．多くの識者の評価や多くの読者の評価が得られます．自己の知識や技術の向上にぜひとも役立ててください．なお，本誌掲載分については，本誌規定の原稿料をお支払いいたします．

プログラム募集のお知らせ

今月号の特集でデバイス・ドライバの作り方は，おわかりいただけたでしょうか．そこで，今回の特集に関連してたとえばXXXXのプリンタをPC-PR201と同じように使えるデバイス・ドライバを作った，画面をCOPYキーでセーブできるTSRを作ったなど，読者のアイディアを活かしたプログラムを募集します．

フロッピ頒布のお知らせ

本誌で掲載したプログラムのソース・リストをフロッピで頒布します．著作権がありますので，コンパイラなどの処理系は読者自身がご用意ください．また，ソース・リストも読者の入力の手間を省くことが主な目的ですので，プログラム上のバグや本誌に掲載した内容以上の質問には応じられません．お申し込みについては本誌157ページをご覧ください．

編集部から

＊トラ技で10年，トラ技スペシャルで2年編集に携わってきました．そして，今回トラ技コンピュータになります．ソフトウェアで技術誌が成り立つのか多少不安に思っていますが，技術は高いところから低いところへ流れるのが当然の姿です．ソフトウェアに関してはまだ一部の特権みたいなところがあるので，少しでも，低いところへ流れるために役立ちたいと思います．〈山〉

＊先日，テレビのカラヤン氏・追悼番組で，初来日の公演の模様が映し出されていました．私が産まれた年のことで，今膝の上にのっている長男は，昨年産まれたばかりです．30年の進歩，とくにここ十年のコンピュータの発展は，情報の伝達と加工などに飛びぬけた進歩が見られます．また，音楽用CDも手軽に扱えるようにはなりましたが，ソースとしての1930年ごろの演奏が心に訴えるものが多いのはどうしてなのでしょうか．そして，なぜヨーロッパのレーベルに感動するものが多いのでしょうか．しかし，あと10年 日本の発展が続けば，その文化も手に入れることができるような気がします．（ヨ）

次号予告

次号は12月18日発売です

パソコンで構築する
◆計測制御ソフトウェアの作成手法

FA/LAと呼ばれる環境でもパソコンがたくさん使われるようになってきました．これは，パソコンを用いる開発環境が整ってきたこと，拡張ボードなどが各種使えて，希望するシステムが容易に組めることなどが大きな理由です．次号では，A-D/D-A，ディジタルI/O，モータ・コントローラなどの拡張ボードを組み合わせ，システムを組むためのノウハウをやさしく説明します．開発用言語としては，Basicを一番多く使いますが，C言語の事例も掲載する予定です．

トラ技コンピュータ　11月号
発行所　CQ出版株式会社　〒170　東京都豊島区巣鴨1-14-2　☎03-947-6311～6315
振　替　東京0-10665
発行人　神戸一夫　編集人　蒲生良治
印刷/製本　三共グラビヤ印刷㈱　表紙デザイン　㈱アイドマ・スタジオ　本文イラスト　神崎真理子
©1989　CQ出版株式会社　（無断転載を禁じます）
1989年11月1日発行
定価は表四に表示してあります

LOTUS、INTEL、MICROSOFT EXPANDED MEMORY SPECIFICATION

LIMEMS

(Lotus1・2・3(ロータス)、一太郎Ver4(ジャストシステム)、EXCEL(マイクロソフト)など、LIM EMS対応アプリケーションが次々と開発されています。)

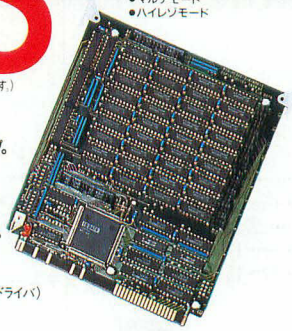
発売以来変らぬ自信の統一規格設計。

LIM4.0EMSをはじめ、メモリ拡張の全モードに対応。共通仕様で1MBから8MBまでフルラインアップ。

EMSプラス増設RAMボード **PIO-PC34H** *Series* (1/2/4/6/8MB)

■対応機種/PC-9800シリーズ全モデル ■価格/¥38,000～¥235,000

PIO-PC34Hシリーズ
- LIM4.0EMSモード
- I・Oバンクモード
- プロテクトモード
- マルチモード
- ハイレゾモード



LIM4.0EMS仕様を完全サポート。さらに80386のプロテクトメモリでEMS、BMSを実現。

EMS/キャッシュ&RAMディスク **IOS-10EMS/386**(新発売)

■機能/EMSドライバ/386メモリドライバ/RAMディスク/ディスクキャッシュ/プリンタスプーラ/バンクドライバ(FEPドライバ)
■価格/¥5,000 バージョンアップサービス¥3,000

LIM EMSの広がりも、マルチウィンドゥのトレンドも。マルチモードだからOK。

EMSプラス増設RAMボード **PIO-AX34E/H** *Series* (2/4/8MB)

■対応機種/PC/ATパソコン※ラップトップ用にハーフサイズもご用意しています。
■価格/¥85,000～¥245,000(LIM4.0EMSに準拠したEMSドライバ5-2DD添付) ※ハーフサイズ 2M/¥87,000 4M/¥149,000

I・O DATA-EXPANDED MEMORY SPECIFICATION

● LIM EMS ASIC(gate array)LSIのOEM供給をいたしております。(EMSドライバソフトも供給可能)
PC34シリーズおよびAX34で使用しているカスタムLSIです。対応モードは ●LIM EMS4.0/3.2をフルサポート ●プロテクトモード ●EMS+プロテクトのマルチモード ●I・Oバンクモード ●ハイレゾモードと全モードに対応、パソコンメーカ様にも続々採用されています。

最初の出逢いからニクイほど変わらない……。
しかも、トレンディ。そんなあなたを選びます。

この、一貫した思想が違いです。

だから、日本の標準。

I・Oバンク方式はもちろん、EMSもまた、発売以来変わらない、同一基本仕様。
ここから生まれる高い汎用性の確保をはじめ、遠い将来も見つめた設計思想を貫いています。

どんなソフトも笑顔で引き受けてくれる……。
そんな、頼れるキミを選びます。

どんなAPソフトにも効果的メモリ環境——。

I・Oバンク方式RAMボードの標準モデル。基本性能に徹した永遠のベストセラー。

I・Oバンク方式 BMS 増設RAMボード **PIO-9234G** *Series* (0.5～16MB)

■対応機種/PC-9801シリーズ全モデル ■価格/¥25,000～¥425,000

1,000日以上のバックアップ。RAMボードの理想を実現。完全自動立上げを実現したプロ仕様の実力派。

I・Oバンク方式 無停電増設RAMボード **PIO-9834L** *Series* (1/2/3/4MB)

■対応機種/PC-9801シリーズ全モデル ■価格/¥90,000～¥315,000(IOS-10SR自動IPソフト添付)

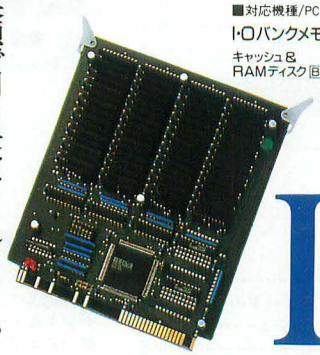
I・Oバンクメモリ/プロテクトメモリの高度活用をはかり、日本語アプリケーションのメモリ環境を大幅に向上。

キャッシュ&RAMディスク BMS **IOS-10**

■機能/BMS/バンクドライバ(FEPドライバ)/RAMディスク/ディスクキャッシュ/EMSエミュレータ/プリンタスプーラ
■価格/¥5,000

I・Oバンクメモリを OS/2メモリ環境でRAMディスク活用!!

OS/2用I・OバンクRAMディスク **IOS-10/OS2**(新発売) ■機能/RAMディスク ■価格/¥18,000



PIO-9234Gシリーズ
- I・Oバンクモード
- 内部メモリ増設

I・O BANK MEMORY SPECIFICATION/I・Oバンクメモリ統一管理規格

I・O BMS

(八方美人2号、UP2シリーズ〈ダイナウェア〉、Recalc〈アスキー〉など、I・O BMS標準搭載アプリケーションが次々と開発されています。)

NECラップトップ機専用メモリ 10月中旬発売	●EMSプラス増設RAMボード PIO-PC34SL(2M)	¥72,000	
	●スタティック 無停電 RAMボード PIO-9834SL(1/2M)	1M/¥90,000 2M/¥165,000	
NEC32ビット機専用メモリ	●RA専用3MB増設RAMボード PIO-RA34-3M 実装済	¥98,000	好評発売中
	●RL専用4MB増設RAMボード PIO-RL34-4M 実装済	¥148,000	
	●ES専用3MB増設RAMボード PIO-ES34-3M 実装済	近日発売	

印刷しながら、次の仕事ができる。

プリンタ・バッファ"PRBシリーズ。"大容量、超小型で新登場。

ハーソナル・プリンタ・バッファ **PRB** *Series* 512/1024(KB)

● PRB-512(0.5MB)…¥39,800 ● PRB-1024(1MB)…¥59,800

データ・セレクタ **PRS-21** 近日発売

I・O DATA パソコン増設RAMのトップブランド 株式会社 アイ・オー・データ機器

本社/サポートセンター 〒920 金沢市駅西本町1-5-41 TEL.0762-21-4812 FAX.0762-24-9300
東京営業所 〒101 東京都千代田区神田富山町6 松崎ビル4F TEL.03-254-0301 FAX.03-254-9609

IOS-10組込み相談室
● 10月21日 11月11日・25日(10:00～15:00)
● 東京営業所
※アプリケーションソフトとIOS-10シリーズソフトを持参下さい
※広告掲載の表示価格には消費税は含まれておりません

トラ技コンピュータ 広告索引

1989.11 Vol.1

五十音別広告索引

ア行

㈱アイ・オー・データ機器……5
㈱アイシーエム……22
アイビックス㈱……表3
岩崎技研工業㈱……17
㈱インタフェース……12～13

カ行

カノープス電子㈱……10～11
関西電機㈱……36
㈱キャラベルデータシステム……19

サ行

㈱サザンパシフィック……14～15
ジョン スワイヤ エンド サンズ ジャパン リミテッド……20
ソニー・テクトロニクス㈱……8～9

タ行

㈱東洋情報システム……31

ナ行

ニコレー・ジャパン㈱……表2
ニシム電子工業㈱……16
㈱日本技術ソフト開発……30
日本システム㈱……29
日本テレコメット㈱……28
日本電気㈱……表4

ハ行

パソコンショップ シグマ……32
㈱ビレッジセンター……26～27

マ行

ミナトエレクトロニクス㈱……21
メガソフト㈱……18
㈱メルコ……7

ヤ行

㈱ユーテック……23

ラ行

㈱ライフボート……24～25

製品別広告索引

半導体

CPU，MPU
日本電気㈱……表4

ボード／モジュール

メモリボード／モジュール
㈱アイ・オー・データ機器……5
㈱メルコ……7
㈱ユーテック……23

その他のディジタル処理用ボード／モジュール
カノープス電子㈱……10～11

通信，伝送用ボード／モジュール
日本テレコメット㈱……28

各種のボード／モジュール
㈱インタフェース……12～13

電源機器

電源装置
ニシム電子工業㈱……16

計測機器

波形計測器
ニコレー・ジャパン㈱……表2

コンピュータ

パーソナルコンピュータ
ジョン スワイヤ エンド サンズ ジャパン リミテッド……20

開発支援機器

インサーキットエミュレータ（ICE）
岩崎技研工業㈱……17

ロジック・アナライザ
ソニー・テクトロニクス㈱……8～9

PROM／PAL書込器
ミナトエレクトロニクス㈱……21

周辺装置

印刷関連機器
関西電機㈱……36

記憶関連機器
㈱アイシーエム……22
アイビックス㈱……表3
㈱キャラベルデータシステム……19

ソフトウェア

ユーティリティソフト
メガソフト㈱……18

エディタソフト
日本システム㈱……29

データベースソフト
㈱サザンパシフィック……14～15

各種のソフトウェア
㈱東洋情報システム……31
㈱ライフボート……24～25

その他のソフトウェア
㈱日本技術ソフト開発……30

流通

各種電子機器・部品販売
パソコンショップ シグマ……32

各種ソフトウェア販売
㈱ビレッジセンター……26

情報，教育

出版，書籍
㈱ビレッジセンター……27

キーワード・アド・インデックス

数字・記号

32ビット＋200MHz……9
80386 33MHz……20
98－FBIシリーズ……12
μ-Series-ROM化開発コンパイラ……25

英字

AD・DA・DI・DOボード……12
Brief……29
C Users Journal Japan……27
dBASEⅢPlus上位互換……14
dBXL……14
EMS……18
EMSボード……7
IBM AT互換……20
KX－3000シリーズ……36
LCDメニュー方式……36
LESAGE……22
MEMORY－PRO386……18
MODEL 1890A/1891……21
MULTIFAX……28
PC－9800シリーズ用ボード……12
PC-98用拡張ディスクシステム……表3
PIO……5
PRISM3000……9
PROICEシリーズ……17
QuickSilver……15
RAMボード……5
STRIDE……22
TIS－システムパッケージ……31
TISソフトウェアプロダクツ……31

ア～カ行

インテリジェントバッファ……36
開発支援ソフト……30
各種ソフトウェア……26
画像処理ボード……23
ギャングプログラマ……21

サ～タ行

シングルチップ・マイコン78K……表4
新装開店パソコンショップ……32
デジタル・オシロスコープ……表2
デバイスドライバ……表3
トライポート方式UPS……16

ハ～マ行

ハードディスク……19
ハードディスク……22
プリンタ・バッファ……5
フレームメモリ……23
メモリー拡張……18
メモリボード……7

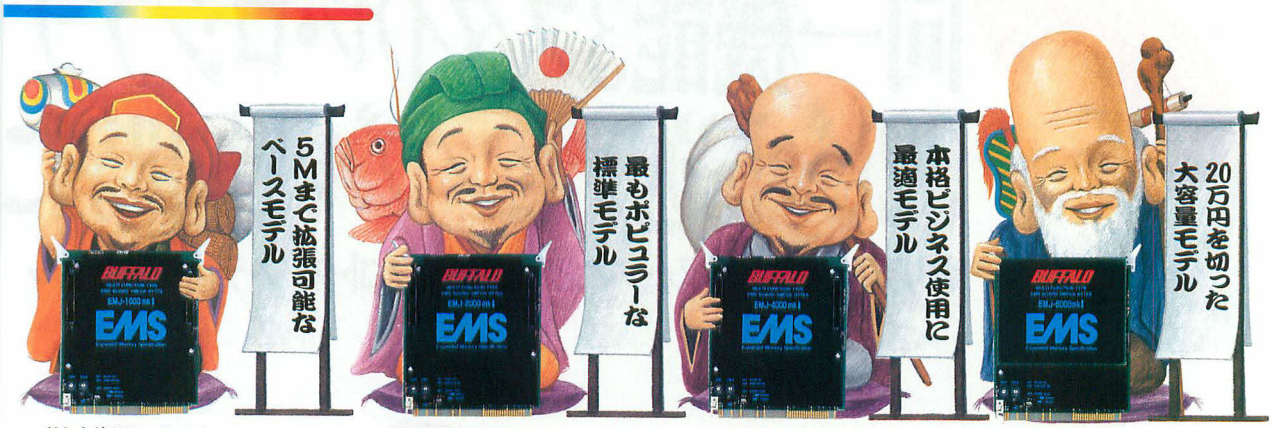
ヤ～ラ行

ユーティリティソフト……30
ユニバーサルプログラマ……21
リレーショナル・データベース……14
ロジック・アナライザ……9

パソコン、もっと使いやすく——

BUFFALO

新発売



1Mバイト(EMJ-1000mkII)　2Mバイト(EMJ-2000mkII)　4Mバイト(EMJ-4000mkII)　6Mバイト(EMJ-6000mkII)

8Mバイト(EMJ-8000mkII)　12Mバイト(EMJ-12MmkII)　16Mバイト(EMJ-16MmkII)

EMJmkIIシリーズ発売記念!

抽選で100名様にモニタープレゼント

- 応募方法:官製ハガキに住所(連絡先)・氏名・年令・職業(職種・会社名・部署)・ご利用のパソコンとソフト名をご記入の上、㈱メルコ本社「EMJmkIIモニタープレゼント」係までお送りください。
- モニター製品:EMJ-1000mkII(1Mバイト)
- 締切:1989年11月10日(必着)
- 発表:主要パソコン雑誌90年1月号(弊社広告欄)にて発表。

EMSボードの真打登場!

快適EMSボードの決定版。こんなお得な話しは、他にない。

1枚のボードで、各種拡張メモリに対応。将来性の豊かな、多機能ボードです。

●ハードウェアEMSメモリ。

ハードウェアによる高速・快適なEMSメモリ拡張を実現。表計算ソフト「1-2-3」「Excel」をはじめ、「一太郎Ver.4」など、EMS対応ソフトの作業領域を飛躍的に拡大します。ページフレームも16ページまで設定でき、1Mバイトまでのメモリ空間において任意のアドレスへ、任意のページ数をオープン可能です。

●プロテクトメモリ、バンク切換メモリをはじめ、本体メモリへの増設も可能です。

※各種の拡張メモリモードの設定は、ボード上のロータリースイッチで切換えることができます。

業界に先がけて、ハイレゾリューションモードにも対応。

RAMウィンドウにより、ハイレゾリューションモードでのプロテクトモード増設を業界で先がけて可能にしました。しかも、EMSメモリとしても活用可能。幅広いユーザーの声にお応えしています。

時代を先行する「MELWARE」。新機能の搭載で、さらに差がでます。

●マルチメモリ・マネージメントシステム〈MMMS〉(トリプル・エム・エス)機能搭載。

EMSメモリ、プロテクトメモリ、バンク切換メモリ、各種拡張メモリをトータルに管理・活用できる先進の〈MMMS〉機能を搭載。メモリの種類にとらわれることなく、各種アプリケーションプログラム空間はもちろん、RAMディスク・キャッシュディスク・プリンタスプーラなどの多彩な機能を、使用環境にあわせて、お望みのメモリ上に、自由に割り振ることが可能です。

●サポートするメモリは、メーカーを選びません。

MELWAREは、BUFFALOの揮発ボードをはじめ、すべてのバンクメモリ(128KB/00ECh)、プロテクトメモリでご使用いただけます。

●80386パソコンではプロテクトメモリを仮想EMSメモリとして利用可能。(MELWARE Ver.4新機能)

386マシンのプロテクトメモリを、オンチップ・ページング機能によりハードウェアEMSメモリと同様の性能で利用できる、新機能が付加されました。

その他充実の先進機能満載。

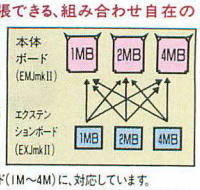
●付属の簡易エディタ(MED)はハイレゾリューションモードでも使用可能なフルスクリーンエディタ。(漢字使用も可能)●簡単設定プログラム、プリントユーティリティー搭載。●キャッシュディスクはSCSI規格ハードディスク対応●NEC拡張ファンクションもサポート●オンラインマニュアルは、画面上・プリントアウトの両方で確認可能。●MS-DOSのBUFFERSの一部をユーザーズメモリ外に追い出せ、ユーザーズメモリの負担をさらに軽減。

- 1Mバイト(EMJ-1000mkII) ……¥37,800
- 2Mバイト(EMJ-2000mkII) ……¥64,800
- 4Mバイト(EMJ-4000mkII) ……¥120,000
- 6Mバイト(EMJ-6000mkII) ……¥175,000
- 8Mバイト(EMJ-8000mkII) ……¥230,000
- 12Mバイト(EMJ-12MmkII) ……¥340,000
- 16Mバイト(EMJ-16MmkII) ……¥450,000

(すべてMELWARE Ver.4標準添付)

オンボード・1スロットで拡張できる、組み合わせ自在のエクステンションボード。

- 1Mバイト(EXJ-1000mkII) ……¥33,000
- 2Mバイト(EXJ-2000mkII) ……¥60,000
- 4Mバイト(EXJ-4000mkII) ……¥114,000



※エクステンションボードは、本体ボード(1M〜4M)に、対応しています。
※サウンドボードを切り離せないUVIIでは、ハードディスクとの併用ができません。
●本製品の価格には消費税が含まれておりません。

データショウ'89に出展
●ブース番号:東館E-21

LIM/EMS 4.0完全準拠・PC-98/9801シリーズ用(全機種・全モード対応)

EMSボード/EMJmkIIシリーズ

株式会社 メルコ

●本社/〒460 名古屋市中区大須4-11-50 カミヤビル
☎(052)251-6891(代)
●東京支店/〒101 東京都千代田区神田須田町2-19-8 酒井ビル
☎(03)255-2247(代)

資料請求No.7

▶お問合せはメルコインフォメーションセンターまで
専用電話 **052-251-8365**
受付:9:30〜12:00/13:00〜17:00(月〜金)

Newロジアナ
32ビット+200MHz

同一機能3スタイル・ロジアナ
32ビット逆アセンブルと同時に

●リアルタイム・パフォーマンス解析●プロトタイプ・デバッグ・ツール

3001MPM型/3001MPX型
64ch/96chステート+200MHz、9chタイミング
ポータブル・ロジック・アナライザ

3002C型
カラーCRTメインフレーム

SONY Tektronix ●常に世界最新を創出するマニファクチャラ

ソニー・テクトロニクス

ソニー・テクトロニクス株式会社
計測機器部／東京都品川区北品川5-9-31 〒141 TEL 03・448・4661
北海道地区代理店／㈱北海道デジタルブレイン TEL011・727・8931
仙台営業所／TEL022・267・2181 土浦営業所／TEL0298・24・2602
大宮営業所／TEL048・646・0711 多摩営業所／TEL0425・73・2111
横浜営業所／TEL045・661・3511 厚木営業所／TEL0462・28・6768
名古屋営業所／TEL052・581・3547 大阪営業所／TEL 06・947・0321
広島地区／TEL082・247・0661 福岡営業所／TEL092・472・2626

ロジック・アナライザをはじめてご使用になる方のための入門書〝ロジック・アナライザ入門〟お送り致します（無料）。ご請求下さい。

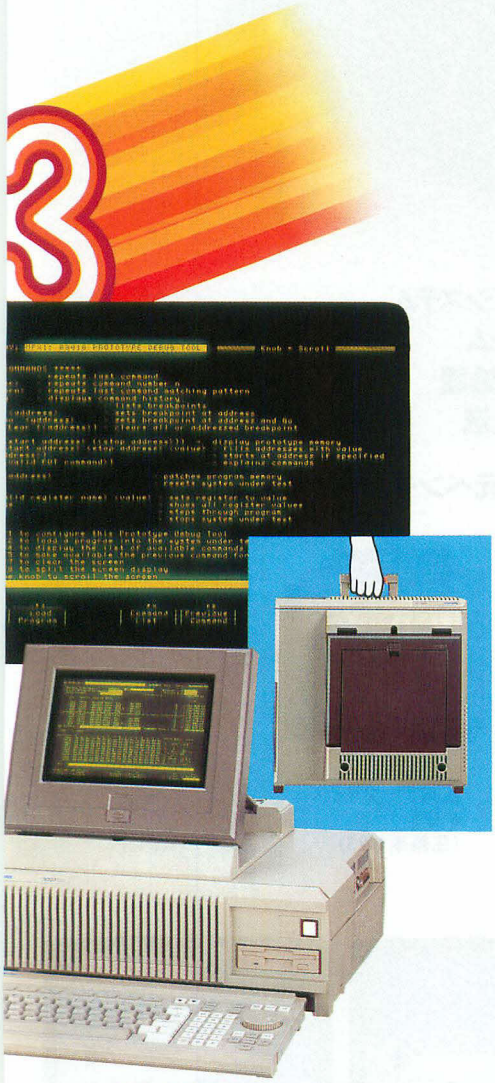
JEMIMA SHOW

'89国際計測工業展
10月24日(火)～27日(金)
東京国際見本市会場
主催：社団法人日本電気計測器工業会
コマ番号 N－95（新館2F）

●セミナー 26(木)B室
11：40～12：40「最新のロジック・アナライザの動向とその応用」

200MHzタイミング解析

●モジュラ構造で最大10スロットまで拡張



3002P型
ELフラット・パネル・ポータブル・メインフレーム

3スタイル

PRISM3000シリーズはポータブル・タイプの3001MPM(64ch)/MPX(96ch)型とカラーCRTで2スロット・メインフレームの3002C型、ELフラット・パネル・2スロット・ポータブル・メインフレーム3002P型の3スタイル4機種です。3スタイルとも同一のユーザ・インタフェースを採用。メインフレームも5台まで増結でき10スロットまで使用できます。

モジュールは現在二つ、新モジュールで将来は機能向上

メインフレーム用のモジュールは64chステート+200MHz、9chタイミングの30MPM型、ステートが96chで他は同一性能の30MPX型の二つです。近い将来に発表予定の新モジュールで一層複雑な問題も効率良く対処できます。

500psチャンネル間スキューで高信頼データ

32ビットまでのプロセッサの逆アセンブルと同時に使える200MHzタイミング解析部はチャンネル間スキュー500ps。6nsの間データが確立すればトリガできる強力なワードリコグナイザは今まで見逃していた事象を確実に補え、しかも全データをステート部と時間的相関を保って表示できます。

解析を容易にするデータ表示フォーマット

ステート解析部のメモリ長はタイムスタンプつきの8Kビット/チャンネル。逆アセンブルされたデータはモード選択によりサブルーチン・コール、リターン、ジャンプ時のバス・サイクルのみを表示できるため、長大なプログラムの流れも簡単にチェックできます。

初のリアルタイム・パフォーマンス解析機能(オプション)

従来の統計的手法では多大の時間を要したり、重要なデータを取り逃すことが避けられなかったパフォーマンス解析を、7個のハードウェアによるレンジ・リコグナイザにより、当社初のリアルタイムの解析を可能にしました。これからは一点の疑いも無いパフォーマンス解析ができます。

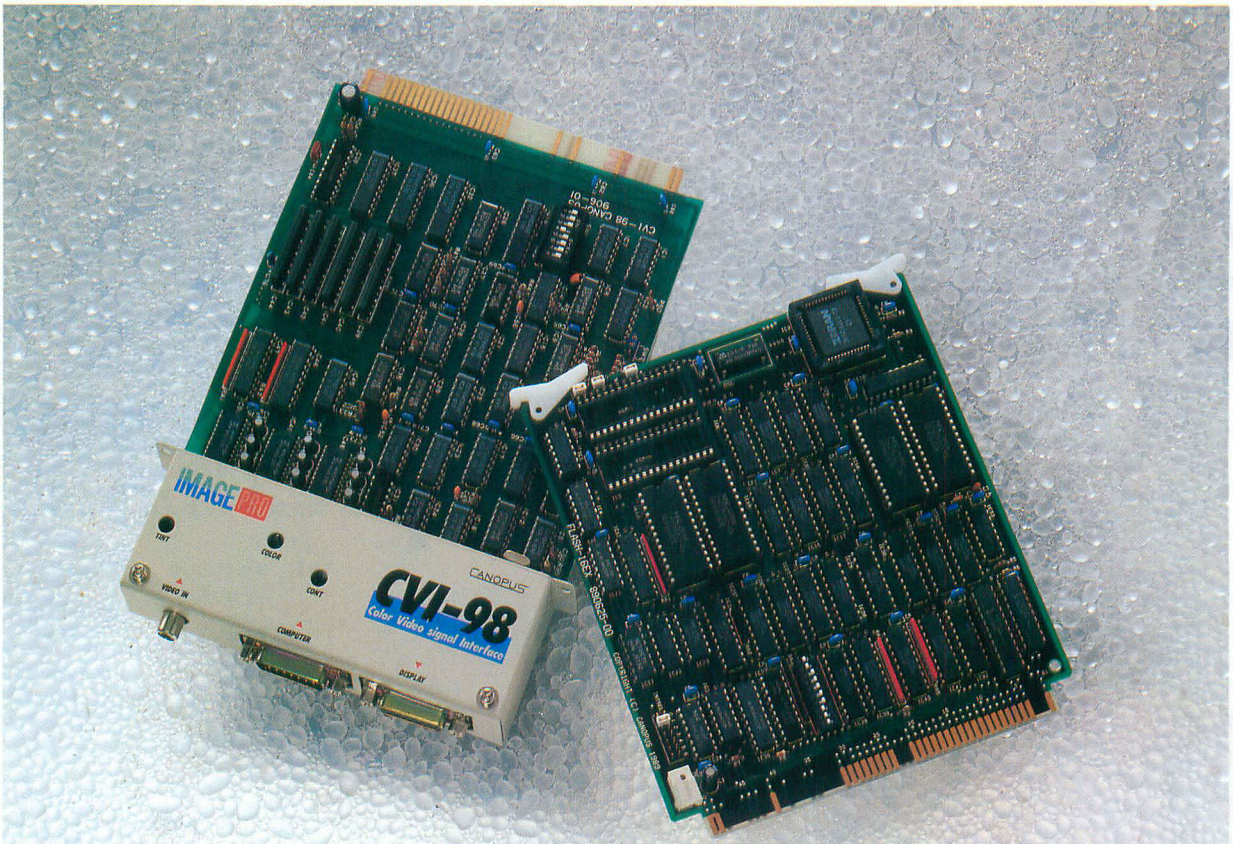
プロトタイプ・デバッグ・ツール(オプション)

プロトタイプ・デバッグ・ツール(PDT)はシステムのROMソケットに差込むROMプローブとデバッグ用ソフトウェアから構成され、簡単なコマンドでできるメモリ・ダンプやパッチの可能なROMエミュレーション機能の他、逆アセンブル・プローブからのハードウェア・ブレーク等によるプロセッサのコントロールができます。

新製品

ロジック・アナライザ

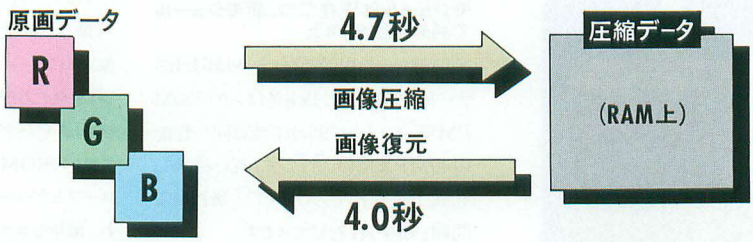
PRISM 3000シリーズ



●画像処理データ圧縮ファイリングシステム
●高速画像データ転送システム
●ネットワーク上での画像データ処理
●パソコン通信での自然画伝送

〔CVI-98付属ソフトウェアを用いた画像圧縮・復元ベンチマークテスト〕

(CVI-98、FLASH-16VIの組み合わせ)

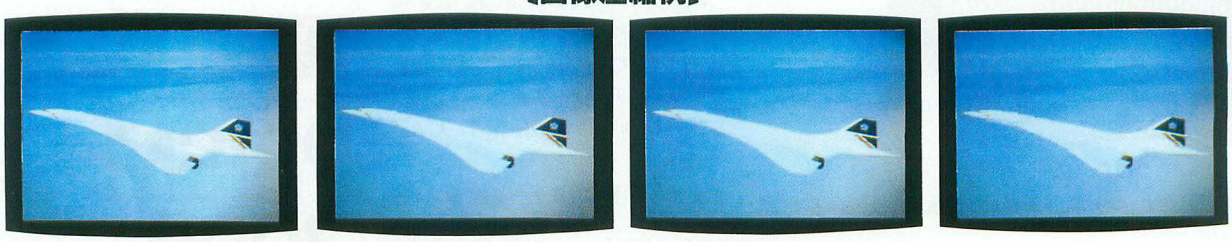


画像データサイズ
150KB
(256×200pixelカラー)

※PC-98RLを使用
下のCONCORDの
絵を圧縮

圧縮データサイズ
6.5KB
(圧縮率1/23)

〔画像圧縮例〕



●原　画　●1/10に圧縮　●1/20に圧縮　●1/25に圧縮

画像処理の分野でも。

CVI-98と、FLASH-16/VIとの組み合わせが画像処理の新分野を開拓。
新たな可能性を秘めたIMAGE PROシリーズにご期待ください。

CVI-98

カラービデオ信号デジタイジングボード

¥98,000

●型番 CIM-1101-01-13××

【CVI-98の概要】

CVI-98は、NEC製パーソナルコンピュータPC-9800シリーズ(LT、CVを除く)の拡張スロットに装着して使用する、ビデオ信号入力・表示ボードです。CVI-98は、入力した標準NTSCビデオ信号から1画面分の画像データを取り込み、記憶(デジタイズ)するためのハードウェアと、記憶した画面をPC-98本体の表示出力に合成表示させる(スーパーインポーズ)ためのハードウェアを内蔵しており、この2つの機能はPC-98本体からI/Oポートを介して容易にコントロールすることができます。
また、取り込んだ画像データも、PC-98本体側よりI/Oポートを介して自由にアクセスできますので、画像データの加工や外部記憶装置への保存等が簡単にできます。さらに、取り込んだ画面の拡大表示やパソコン画面との合成表示のON、OFFコントロールが可能です。このようにCVI-98は画像処理分野には不可欠なハードウェアであると言えます。

【CVI-98の仕様】

- デジタイズ用映像入力…標準NTSCビデオ信号(ピンジャック)
- スーパーインポーズ用映像入力…アナログRGB信号(15ピンコネクタ・ピンタイプ)
- 映像出力……アナログRGB信号(15ピンコネクタ・ジャックタイプ)
- デジタイズ画面の構成 ‥256×200画素
  RGB各64レベル(6ビット)
  262、144色表示
- デジタイズ画面入力時間…1画面/フィールド
- I/Oインターフェイス ……I/Oアドレスは16ビットデコード
- 使用I/Oポート数 ……入力1ポート(××D2)
  出力4ポート(××D0、××D1、
  ××D2、××D4)
  ××はディップスイッチで選択可能
- 適合コンピュータ……日本電気㈱製
  PC-9801/E/F/M/VF/U/VM/UV/VX/UX/RA/RX/ES/EX
  PC-98XA/XL/XL²/RL
  FC-9801/V/X
  EPSON㈱製
  PC-286/V/VE/VS/VF/X/U/US/LS
  PC-386/V/LS
- 適合ディスプレイ……アナログRGB入力端子付き高解像度ディスプレイ(水平周波数24kHz)
- 消費電流……+5V 500mA +12V 80mA
- 基板サイズ……148×220㎜
  (装着時スロットから53㎜飛び出します。)
- その他の機能 ……デジタイズ画面拡大表示機能
  スーパーインポーズのON、OFFコントロール機能
  デジタイズ画面の表示位置設定(2ヵ所)
  デジタイズ画面データをPC-98本体側からアクセスする機能

【CVI-98付属ソフトウェア】

CVI-98にはすぐに使えるサンプルソフト(ソース付き)が付属しています。画像データを取り込んで圧縮したり、画像ファイルを呼び出したりできます。
また、各言語からサブルーチンのように使える画像圧縮ライブラリ(CVI-98専用)が付属しています。

【CVI-98ブロック図】

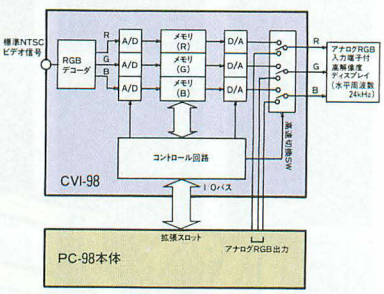


IMAGE PRO

FLASH-16VI

ハイコストパフォーマンス高速演算ボード

¥198,000

●型番 CAU-1106-01-13××

【FLASH-16VIの概要】

ディジタル信号処理にあける画期的なスピードと使いやすさを実現した「FLASH-16」。高速FFT演算ボードとして多くのユーザにお使い頂いています。このFLASH-16で培われた高速性、高機能をそのままに、画像圧縮のためのFCT演算専用にシェイプアップし、ハイコストパフォーマンスを追及して誕生したのが「FLASH-16VI」。FCTによる画像圧縮をFLASH-16VIのアレイプロセッサVSPが高速実行します。

【FLASH-16VIの仕様】

- ホストコンピュータ……全PC-9800シリーズ(LTは除く)
- DSP……ZR34161-20(Vector Signal Prosessor)
- RAM……32KW×2ブロック
  パイプライン処理をサポート(注1)
  (64KW1ブロックとしても動作可能)
- ROM……無(ソケットのみ実装)(注2)
- PC側使用メモリ空間…8KB(000000～0FFFFF 内の任意のアドレスにショートプラグで設定可能)(注3)
- PC側使用I/O空間 ‥‥無
- 98拡張インタラプトライン…0～6をショートプラグで選択
- 消費電力……動作時最大240mA
- 演算方式……16ビット・ブロックフローティングポイント
  16ビット整数(VSP内部では19ビット)
- データ転送方式……
  1. VSPローカルRAMをPCよりダイレクトにリードライト
  2. 32KW×2の2ブロックモードではVSPがアクセスしていない方のRAMブロックにPCからデータの転送が自由にできます。(高速パイプライン処理可能)

(注1、注2) RAM、ROMはモード設定によりコード、データエリアとしての割り当て方式が変更されます。
(注3) FLASH-16VIのアドレス設定を全て異なる値にしておけば複数をスロットにセットできます。

CANOPUS

カノープス電子㈱
本社〒658神戸市東灘区西岡本1丁目4-30カノープスビル/ご注文・納期のお問い合わせは〈営業部〉TEL:078(411)5292 Fax:078(431)7610
技術的なお問い合わせは〈テクニカル インフォメーション〉TEL:078(412)7166(月曜～金曜/PM1:00～4:00) Fax:078(411)5084

CPU
メモリ
98・FBI series
Factory Bus Interface
NECパーソナルコンピュータPC-9800・FC-9800シリーズ用
DI
DO
AD
DI・DO
パソコンシステムの
機能UP!!
多層基板採用

広島
〒732 広島市南区大州5丁目8-26
TEL(082)281-7777
FAX(082)282-8360
担当営業　平本

大阪
〒542 大阪市中央区上汐2丁目2-11
TEL(06)763-4440
FAX(06)763-4450
担当営業　細田、渡部

東京
〒105 東京都港区芝3丁目31-8
TEL(03)454-4644
FAX(03)454-8137
担当営業　鎌田、山賀、日高

Interface
Dream & Freedom
株式会社 インタフェース

dBXL™ より高速にアップデート

使いやすさを追求した、低価格、高機能リレーショナル・データベース

QUICKSILVER™

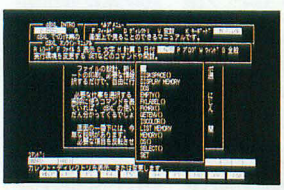
データベース・コンパイラ

はじめて使う人にINTROコマンド

プルダウンメニューに従い、項目を選択するだけで、データベースを作り、処理することができます。表形式でデータを表示させながら、レコードの追加、編集、削除、検索、そしてラベルやレポートの印刷も可能です。プログラムを作らなくても、INTROで行ったことをマクロで登録できるので、同じメニューをなんども開く煩わしさはありません。

オンラインヘルプ機能

コマンドや関数の使い方などマニュアルのほとんどの部分はオンラインマニュアルとしてヘルプキーによって、いつでもどんなところからもポップアップして画面で見ることができます。また、データベース・ファイルのフィールド名の一覧を表示してコマンドの中に自動的に書き込みも可能です。

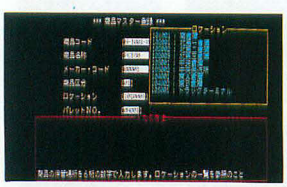


データとプログラムの上位互換性

dBASEIII及びdBASEIIIPlusのデータベース・ファイルやインデックス・ファイル、プログラム・ファイル、ラベルやレポート・ファイルなどと互換性がありますので、dBASEで利用してきたデータやプログラムを有効に活用。ロータス1-2-3やマルチプランといった表計算ソフトのデータファイルの読み書き。もちろん、ロータスやマルチプランの日本独自の形式もサポート。ワープロソフトなどで作ったテキストファイルをデータベースに。また、逆にテキストファイルを出力することもできます。

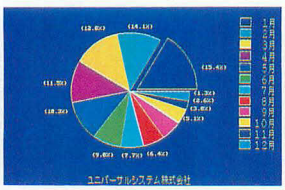
ウインドウ機能

ウィンドウを使ったプログラムを書くことは大変なことですが、dBXLではデータベースを扱うのと同じような感覚で簡単に書くことができます。dBXLでは理論的には99個までのポップアップ・ウィンドウを重ね合わせて開くことができます。それぞれのウインドウで異なるデータを処理したり、メッセージを表示したり。ユーザーインターフェイスに優れたアプリケーションを簡単に作れます。



ビジネスグラフ機能

数字だけ並べたデータではとらえにくい特徴も、グラフを使って表示すれば一目瞭然です。グラフは全部で6種類(円グラフ、棒グラフ、折れ線グラフ、点グラフ、階層グラフ、回帰直線)。ビジネス分野で威力を発揮。
簡単な操作で表現力に優れた資料を簡単に作り出すことができます。もちろんこのデータをプリンターに出力することもできます。

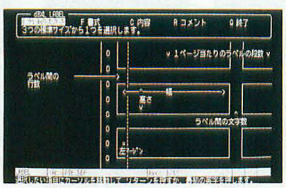


インデックスファイルによる高速検索

dBASEIIIPlusより高速化されたインデックスを使った高速検索が可能。このインデックスを利用して、条件にあう項目だけの書き換えや2つのデータベースを接続することもでき。多重検索や一定の条件にあったデータベース・ファイルを作ることも可能。

ラベルやレポートのプリントアウト

郵送用の宛名印刷や宅急便の伝票ラベルそして業務報告書などのレポートを画面の表示に従って自由なフォーマットでプリントアウト。印刷項目はデータベースの中から選択でき、また出力するデータを条件づけ、選択することもできます。



イメージデータの利用

フルカラーのイメージデータを、簡単に利用することが出来ます。画像ファイリングシステム(画像データベース)を簡単につくれます。ユーティリティを利用して・イメージスキャナから取り込んだデータを利用することも可能。
dBXLで直接イメージスキャナから画像を読みとることはできません。画像の取り込みにはスキャーナに付属のソフトウエアなどをご使用ください。

EMSのサポート

EMS(LIM仕様 Ver4.0)に対応していますので、市販されているEMSボードを利用して使えるメモリを増やすことができます。
この機能はNEC方式のEMSでは動作しません。また、一部メーカーのEMSボードやソフトウエアの組合せでは動作しないことがあります。

新しく追加された機能

dBXLのなかから、ワープロや、表計算などを実行する時、SET SWAPPING ONを設定するとdBXLは、自分自身をdisk上に書き出しメモリーを解放。複数同時リレーション機能で元データベースから複数のデータ・ファイルを同時に関連付。TSRタイプのホット・キーでプログラム実行中に32個までの別のプログラムを起動。不可能だった可変長メモ・フィールドを自由に検索したり置換。

便利なユーティリティ

画面のイメージを、dBXLの扱えるファイル形式に変換するユティリテー、漢字フロントプロッセサ(ATOK6)をグラッフィクメモリに移動してユーザーメモリを広げるデバイスドライバー、dBXLのサンプルプログラムとして、そのまま使えるカード型データベースThe FlatFileなどが新しく追加されました。

MS-DOS汎用版

dBASEIIIPlusの対応していない機種でも、MS-DOS(V.2.11以降)が走っている機種であれば、どのマシンでも動作します。これはグラフの機能がないだけで、データベース・ファイルやインデックス・ファイル、プログラム・ファイルといったファイル関係では互換性があります。なおMS-DOSの標準入出力を使用している関係上、専用版より画面表示は遅くなります。

dBXL
PC-286/PC-9801 …… ¥49,800
J-3100 …… ¥49,800
AX …… ¥49,800
PS/55 …… ¥49,800
MS-DOS汎用 …… ¥49,800
IBM PC …… ¥49,800

ご注文の際は機種、ディスクのフォーマットをご指定ください。
消費税3%が別にかかります

各々の商品名はそれぞれの会社の登録商標もしくは商標です。

高速化

QuickSilver V1.3では、PC-286/PC-9801, J-3100, AX, PS/55シリーズ用に、新しく機種専用版が発売になりました。7倍以上高速になり、カラーをサポートするなどdBXL互換のコンソール制御が実現されました。そして機種対応の結果、ファンクション・キーやカーソル・キー、COMポートがサポートされました。また、レコード・ロック、ファイル・ロックなどだけを使ったシンプルなネットワーク・システムには、LAN Kitは不要になりました。

QuickSilverは、世界の標準リレーショナル・データベースであるdBASEIIIPlus や、その拡張互換のdBXLのADL(アプリケーション開発言語)で作成したプログラムをコンパイルして、実行形式のEXEファイルを生成します。できあがったプログラムはネイティブコードですので、よりコンパクトでより高速で動きます。このプログラムを実行するにはdBASEIIIPlusもdBXLもいりません。

優れた開発環境

QuickSilverとdBXLは、dBASE言語でのアプリケーション・プログラムの開発に関して、他のいかなる言語よりも効率のよい開発環境を提供します。dBASEIIIPlusと文法、データ、インデックス・ファイルも完全に互換性がありそのうえ、dBASEIIIPlusの機能をより使いやすく拡張しています。

ネットワーク

ローカルエリアネットワーク上では、dBASE IIIPlus LAN、dBXL/LAN、QuickSilverのプログラムで同一のデータを共有できます。コンパイルしたプログラムは社内で配布したり、業務ソフトとして販売してもライセンス料がかかりません。
なおマイクロ/メインフレームブリッジSQLを89年末から90年にはSQL対応日本語版として出荷予定です。89年10月にはUnix版(英語)を発売する予定です。
またMS汎用版はdBASEIIIPlusの走らない機種でも使用することができます。

QuickSilver
PC-9801/PC-286 …… ¥165,000
J-3100 …… ¥165,000
AX …… ¥165,000
PS/55 …… ¥165,000
MS-DOS汎用 …… ¥145,000
IBM PC …… ¥135,000
QuickSilver登録ユーザーのみ
Plink86Plus …… ¥80,000
別売ライブラリ …… 各 ¥45,000
4本セット ¥135,000
LAN Kit …… サーバ毎 ¥35,000
Networker Plus
…… サーバ毎 ¥58,000
(PC-DOS専用)

ドレッシングの選択をまちがえても次がありますが...
データベースは貴重なデータが使えなくなります。

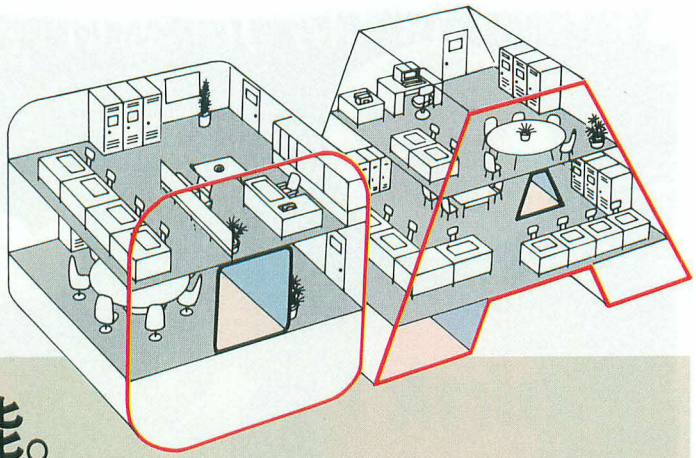
1つのファイルしか扱えないカード型、互換性のないデータベース、ビジネスの世界ではもう使えません。時代はリレーショナル。複雑なビジネス社会のパワーツール、dBXL。それでもまだ、カード型データベースを使いたいと考えているあなたのため。 dBXLで書かれたカード型データベースThe FlatFileが付属します。データフォーマットは、世界中でリレーショナル・データベース標準のdBASEIIIPlusと完全に互換。豊富なサードパーティのアドオンソフトでいろいろな世界が展開されています。dBXLは、初めてデータベースを使う人から業務ユーザーまで、dBASEIIIPlus拡張互換。米国では、社会保障局の標準データベースとして、全米各地の同事務所で約1万本が使われています。初めての人には使いやすいINTROコマンド。業務ユーザーには強力なオンラインヘルプの下で、高速なインデックス検索、自由自在の多重検索、ウィンドウ、グラフ、ラベル・プリント、画像データなど。そしてADL（内蔵のアプリケーション開発言語）によるプログラムで、売上管理、在庫管理、財務会計などあらゆる業務の処理が可能になります。dBASEIIIPlusのADLで開発されたほとんどのプログラム・ファイルは変更なく実行することができます。そしてdBASEIIIPlusと互換のデータ・フォーマット。ロータス1-2-3やマルチプランのデータも自由に使うことができます。使いやすさを追求したユーザー・インターフェイス。データはソフトウェアより貴重なものです。一度入力されたデータは使えば使うほど大きくなります。一つのデータをロータス1-2-3で、マルチプランで、共有。ワープロで入力されたテキスト・フォーマットからdBXLのデータ・フォーマットへそのまま変換。対応機種はIBM PC、PC-286/PC-9801、J3100、AX、PS/55。 そしてdBASEIIIPlusの走らない機種でも使用することができるMS-DOS汎用版も。ローカル・エリア・ネットワーク、dBXL/LAN、QuickSilver(ADLコンパイラ)を使ってデータをオフィスで共有できます。今後対応機種もワークステーションまで増え、データを自由に別の機種で利用可能。パーソナルなデータベースから世界的な大規模データベースまでdBXL、QuickSilverは対応しています。QuickSilverは業務ソフト開発に国内数千本の実績。簡単な住所録から国際的企業の在庫管理、会計処理、業務処理プログラムなど広く使われています。初めて使う人は対話型コマンドで、INTROコマンドで、使いなれてきたらADL（アプリケーション開発言語）で。パワー・プログラマ向けのコマンドもサポート。LANを使ってマルチユーザー企業内システムに。イメージ・データを使った画像ファイリング・システムに、グラフ機能を使ったプレゼンテーションに、ラベル機能でメーリング宅急便に、レポート機能で報告書作成に、いろいろな分野で活躍しています。使えば使うほどその実力がわかります。dBASEIIIPlusをすでに使っている方にとってもより強力なプログラミングツールとして二度と手放せないものとなるでしょう。 それとも、まだdBASEIVを待ちますか。使い方がわからなくなった時にわかりやすいマニュアルが、初めてデータベースを使う人に学習編とINTRO、どこからでも呼べるウィンドウを使ったヘルプ機能、エラーには的確なエラー・メッセージ、そしてプログラム開発には使いやすいエディタ、実行エラーには強力なFixオプションでエラーの箇所を指摘。 マニュアルは約1000ページ。ヘルプ画面はマニュアルの1/3以上。より詳しく知りたいときだけ、マニュアルはお読み下さい。dBXLユーザー・クラブに参加して情報交換。VARの方はQ.S.VAR会へ。サザンパシフィックとWordTech社により情報提供。もちろん当社のテック・サポート体制も充分。

ーそれでもdBXL、価格は¥49,800。ー

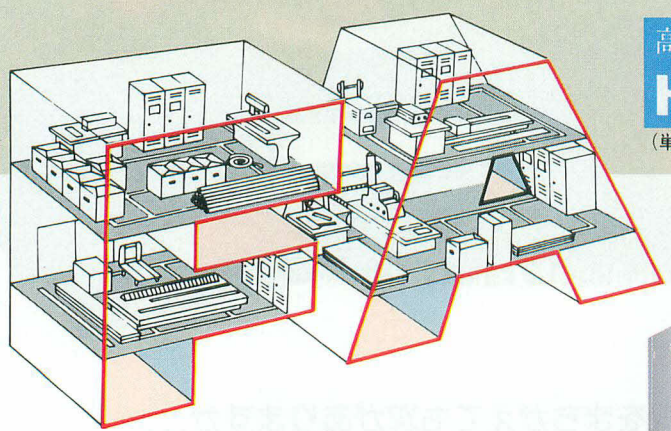
お問い合わせは
全国パソコンショップまたは
㈱サザンパシフィック

〒220横浜市西区南幸2-16-20　三和横浜ビル
TEL.045-314-9514　FAX.045-314-9840

交流無停電電源装置

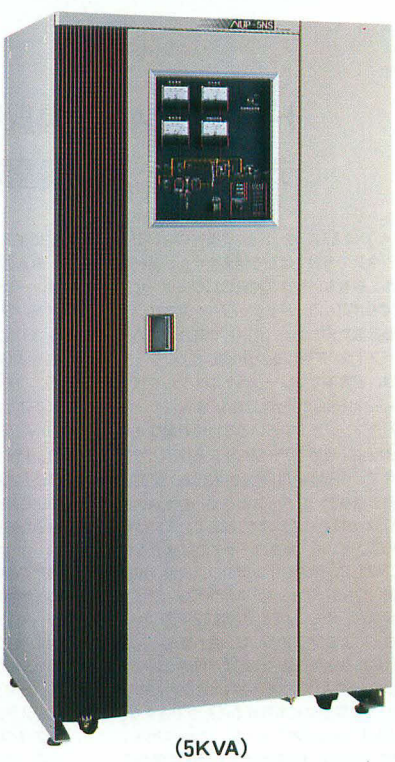


先進、高性能。
さらに安心のワークエリア。



高効率・高経済性・高信頼性の
トライポート方式UPS
（単相3KVA～100KVA：3相10KVA～300KVA）

'89省エネルギー優秀製品賞受賞



(5KVA)

新感覚デザインの実力派

コンピュータや電子応用機器などの重要機器用電源装置には、従来の無停電・定電圧・定周波特性に加えて、高効率・高経済性・高信頼性を兼ね備えた性能・機能が要求されております。これらのニーズにお応えするため当社は、AT & T社（米国電話電信会社）の特許（トライポート方式）の実施権を得、さらに当社開発の新技術（特許出願中）を加えて、一歩先進の省エネ型トライポート方式UPSを完成させました。あらゆる電源トラブルに即応、安定した電力供給に実力を発揮します。

さらに充実した機能設計

- ワンタッチ操作のシステム起動・停止スイッチを装備
- オフィス内設置にも最適なハイセンスデザイン
- 小型軽量化を実現
- オートリターン機能付

ニシム電子工業株式会社
NISHIMU ELECTRONICS INDUSTRIES CO. LTD.
本社/福岡市中央区渡辺通2丁目1番82号（電気ビル内）営業所/東京・大阪・福岡・北九州・佐賀・長崎・大分・熊本・宮崎・鹿児島
●お問合わせ/本社営業部☎092(761)0636/東京営業所☎03(452)4471/大阪営業所☎06(535)5992

SOFTWARE REVOLUTION

IWASAKI

まず、サポートリストでご確認ください。

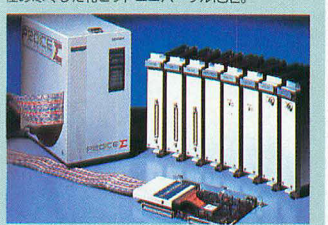
ソフトウェア開発に求められる最もベストな開発環境。それはソフトとハードにおけるトータルな支援体制によって決まります。そのためには開発ホストとICEを接なく先進のアーキテクチャ、ICEそれ自体のもつハイパフォーマンス、それらの機能を100パーセント、フルに引き出すソフトウェアの独自開発が絶対条件となります。PROICEは、当初よりこのコンセプトを徹底して貫き、8ビットから16ビット、そして32ビット時代に向けて〈ザ・ベスト〉を提供し続けます。

高性能ユニバーサルインサーキットエミュレータ
PROICE Σ32

モトローラ32ビットMPU68020をサポート／
25MHzの高速クロックに対応／

UNIVERSAL IN-CIRCUIT EMULATOR
PROICE Σ

独自HLLD搭載。C言語レベルでの高能率環境を極め尽くした16ビットユニバーサルICE。



8 BIT IN-CIRCUIT EMULATOR
PROICE AC

専用パラレル、オン・ザ・フライ、クロスアセンブラ、理想の8ビット開発環境を提供。

IWASAKI DEVELOPMENT SYSTEM SUPPORT LIST

CPU	商品名	カバレッジ パフォーマンス	HLLD	アセンブラ	コンパイラ
Z80	PROICE-Z80/PC	×	▶ ――	▶ PROASM-Ⅱ	▶ LSI C/μC-Z80
64180Z 64180R0/R1	PROICE-180/PC	×			▶ MCC180 μC-180
	PROICE-180A/PC	○			
8085	PROICE-85/PC	×			▶ LSI C
8051/31	PROICE-51/PC	○			▶ μC-8051
8048/49/40/35/39	PROICE-48/PC	×			▶ ×
NSC-800	PROICE-N800/PC	×			▶ ×
6301V/03R	PROICE-01V/PC	×			▶ μC-6301
6301X/Y.03X/Y	PROICE-01X/PC	×			
	PROICE-01XA/PC	○			
6809/09E	PROICE-09/PC	×			▶ ×
647180	PROICE/PC²647180	△			▶ MCC180 μC-180
64180S	PROICE/PC²64180S*1	△			
68HC11	PROICE/PC²68HC11	△			▶ ×
8086/88	PROICE-86/PC	×		▶ (コンパイラに含む)	▶ Aztec C86
68000	PROICE-68K/PC	×			▶ MCC68K
H16	PROICE/Σ16	△	ソースレベルデバッガ iDEB ▶ ソースレベルビジュアルデバッガ iDEB-Ⅱ (オプション)		▶ 日立64106 Cコンパイラ Whitesmith C
H8/532	PROICE/ΣH8/500	△		▶ PROASM-Ⅱ	▶ 日立647008 C コンパイラ
H8/330	PROICE/ΣH8/300*1	△			▶ ×
8086/88	PROICE/Σ86	△		▶ MASM (コンパイラに含む)	▶ Aztec C86 MS-C Lattice C SSI ROMable C Whitesmith C TURBO C*2 PL/M-86*2
80186/188	PROICE/Σ186	△			
80C186/C188	PROICE/ΣC186	△			
V20/V30	PROICE/ΣV20	△			
V25/V35	PROICE/ΣV25	△			
V40/V50	PROICE/ΣV40	△			
68000/10	PROICE/Σ68K	△		▶ (コンパイラに含む)	▶ MCC68K Whitsmith C
68020	PROICE/Σ³² 68020	△			

Note○：サポート △：オプション ×：不可 *1開発中 *2IDEBⅡのみ
注）シンボリックデバッグは全商品サポート（但し「PROICE-48/PC」のみサポート不可）

技術的なお問合わせは
新設のサポートコールへ

フリーダイヤル（料金無料）
0120-10-7766

（従来のサポートセンターへのお問合わせもすべてこのコールで承ります。）

IN-CIRCUIT EMULATOR
PROICEシリーズ

IWASAKI 岩崎技研工業株式会社

●本社 〒612京都市伏見区下鳥羽平塚町13番地3 ☎(075)602-7878㈹ ●東京営業所 ☎(03)340-2901㈹ ●横浜営業所 ☎(045)201-6631㈹
●大阪営業所 ☎(06)358-5571㈹ ●九州営業所 ☎(092)483-0071㈹

MEGASOFT

386ユーザーの選択

ハードウェアEMSボードが必要とされたのはもう過去の話。スピード・価格・柔軟性などあらゆる性能でハードウェアEMSを上回る仮想EMS方式。もしあなたが386パソコンをお持ちならMEMORY-PRO386が最良の選択です。

主な特長

❶ハードウェアEMSより高速 PC-9801RAやRLなど32ビットパソコンで使えば市販EMSボードより高速です。

❷1万円で実現できるEMS 386パソコンに標準装備の内蔵メモリを使えますから、ソフトウェアの購入だけでEMSを実現します。

❸一太郎 Ver.4にも完全対応 アプリケーションから見れば仮想EMSはハードウェアEMSとまったく同じ。一太郎のフル機能を引き出します。

❹バンクメモリボード対応ソフトも実行可能 仮想バンクメモリ機構でMX-1Starなどバンクメモリ対応ソフトも実行できます。もちろん、バンクメモリボードとの混在も可能です。

❺プロテクトモードメモリだけを使いますから、OS/2実行時にも、ボードの設定を変更する手間は不要。 メモリを最大限有効に活用できます。

❻最大128Kバイトのユーザー空間を拡張 仮想UMB機能によって、たとえば松茸V2を組み込んだ状態でも580Kバイト以上のユーザーメモリを確保。ロータス1-2-3をはじめ、LANやCADソフトなどユーザー空間が不足しやすいソフトも余裕を持って実行できます。

❼PC-9801LSでも使えます 内部増設メモリだけで使えますからPC-9801LSなど拡張スロットのない機種でもOK。

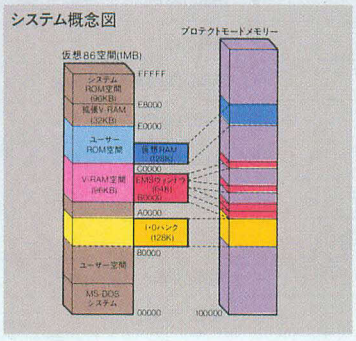
❽どのメーカーのボードも使えます メモリーボードをプロテクトモードに設定すれば、異なるメーカーのボードの混在も可能。安心して増設が行なえます。

❾これからの主流です 仮想EMSは、386CPUのもつ優れた能力を素直に引き出すことで実現した方式。新たなハードが不要で信頼性やコストパフォーマンスも抜群。

❿再購入せずに機能追加が可能 すべての機能をソフトウェアで実現していますから、将来の機能アップもソフトのバージョンアップで可能。ハードを無駄にすることはありません。

●仮想86モードで動作するため、80386/80386SX以外のCPUでは使用できません。

システム概念図



EMS対応ソフトウェア

- 一太郎 Ver.4(ジャストシステム)
- 1-2-3(ロータス)
- EXCEL(マイクロソフト)
- AUTOCAD(オートデスク)
- 花子 Ver.2(ジャストシステム)(予定)
- Advanced BITS 5.0(イシガキMES)
- ターボC(ボーランドジャパン)
- VJE-β Ver.2. 10
- ATOK-7(一太郎に付属)
- 松茸2.05(ロータス1-2-3に付属)

など(弊社確認分)

仮想UMBに組み込み可能なFEPやドライバ

- VJE-β
- ATOK 7
- 松茸/V2
- E1
- MX-1Star
- SC など

使用可能機種

- PC-9801RA/ES/LS, PC-98RL/XL²
- PC-386

使用可能メモリボード

各社のプロテクトモードメモリ
RA/RL/LS/XL²/PC-386本体内蔵メモリ/増設メモリ

日本電気 PC-9801-51/52/54
I・Oデータ機器 PIO-9X34/P, PIO-9234Pシリーズ
メルコ HB/HC/HCE/SB(mk-IIを除く)/XB/XC/EMAシリーズ
(いずれもプロテクトモードで使用)

製品の内容

5"2HDまたは3.5"2HDフロッピーディスク……1枚
取扱説明書……1部
ユーザー登録カード……1枚

ロータス1-2-3での複写速度
1セル20文字のデータを10桁×4000行複写時の所要時間を測定

ロータス1-2-3でのメモリ拡張の効果
1セル20文字のデータを最大いくらまでコピーできるかを測定

ユーザーメモリ:コマンドラインでCHKDSK実行時に表示される使用可能メモリの表示
最大有効セル数:1セル20文字のデータを/Cコマンドでコピーできる最大セル数
テスト環境
使用機種 PC-9801RA 5
増設メモリ RA用内部増設メモリ3MB+ハードウェアEMSボード(2MB)を増設
組み込みドライバ 各EMSドライバ, PRINT.SYS, RSDRV.SYS, MTTKZ.DRV

これからの主流。

仮想EMSドライバ
MEMORY-PRO386

PC-9801RA/LS/PC-98RL/XL²/PC-386専用…¥10,000

●表示価格には消費税は含まれておりません。

使いやすさと信頼性
メガソフト株式会社
〒564 大阪府吹田市江の木町16-9 近藤ビル6F
TEL. 06-386-2058(代) FAX. 06-386-2123

資料請求No. 18

フロッピーディスクの使い方を変えるニューツール!

フロッピーディスク・デュプリケーション・コンバータ

MDP-6401C 3.5″↔3.5″ 定価 ¥188,000

MDP-6402C 5″↔3.5″ 定価 ¥188,000

MDP-6403C 5″↔5″ 定価 ¥188,000

- 3.5″, 5″, フロッピーディスクのメディアチェンジおよびデュプリケートが簡単にできます。
- 2HD, 2DD両タイプに対応。しかもこれらを自動識別します。
- FDから読み込んだデータをそのまま複製するモードと、データを一部解析, 修正して複製するモードがあります。
- NEC PC-9800シリーズの外部拡張ディスクとしても使用可能。

新発売

【ご注意】許可なく他人の著作物を複製(コピー)する事は法律で禁じられております。

高性能FDDシステム同時新発売!

NEC PC-9801用 フロッピーディスクシステム

CA-302FW 3.5″×2 定価 ¥67,800

CA-502FW 5″×2 定価 ¥67,800

新発売

- 2DD/2HD選択可能
- ドライブセレクト(1, 2または3, 4)が自由に設定できます。

＊2HD, 2DD用ケーブル別売

日頃のご愛顧に感謝をこめて 新価格 プライスダウン!

新価格!! 新シリーズ登場でさらにお求め易くなりました。
高速、大容量、使い易くて信頼できるキャラベルのハードディスク群。

80MB単一でブートアップ.!!

NEC PC-9800シリーズ用ミニタイプハードディスクシステム

CA-80LG 80MB 18msec.以下 新価格 ¥218,000 ●I/Fカード, ケーブル付

遂に達成.!!40MBで18m sec..!!

NEC PC-9800シリーズ用ミニタイプハードディスクシステム

CA-44LG 40MB 18msec.以下 新価格 ¥148,000 ●I/Fカード, ケーブル付

よりお求め易くなりました.!!

NEC PC-9800シリーズ用ミニタイプハードディスクシステム

CA-40LG 40MB 35msec.以下 新価格 ¥118,000 ●I/Fカード, ケーブル付

MS-DOS、新松etc.…ソフト内蔵.!!

NEC PC-9800シリーズ用
ミニタイプハードディスク&ソフトウェアシステム

CA-80LGS 80MB 18msec.以下 新価格 ¥258,000 ●I/Fカード, ケーブル付

CA-40LGS 40MB 35msec.以下 新価格 ¥168,000 ●I/Fカード, ケーブル付

【搭載ソフトウェア】●日本語MS-DOS(Ver. 3.1)アプリケーションソフト実行セットPS98-012-HMW一式〈日本電気㈱製〉●日本語ワードプロセッサ「新松」パッケージ一式〈㈱管理工学研究所製〉●アドレス帳(データ100名限定仕様) ●グラフィックデータ集 ＊ソフトウェアはすべて通常ご購入の場合と同様の保証、サービスが受けられます。

内蔵用も高速・高信頼性.!!

NEC-9801RA2/RX2/VM11用 内蔵ハードディスク

CA-818 80MB 18msec.以下 新価格 ¥218,000

CA-428 40MB 28msec.以下 新価格 ¥148,000

増設専用内蔵ハードディスク

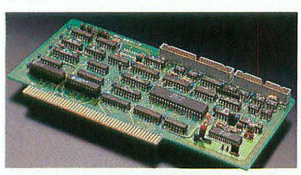
CA-428+ 40MB 28msec.以下 定価 ¥138,000

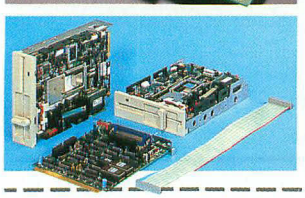
Mac用も高速・高信頼性.!!

Macintosh II/SE用
内蔵型ハードディスクシステム

EG-80II/SE 80MB 18msec.以下 新価格 ¥188,000

EG-40II/SE 40MB 28msec.以下 新価格 ¥128,000





活用多彩、充実のI/Fカード群!!

98VFをVM仕様に.!!

フロッピーマルチ I/Fカード **PC-98M17** 定価¥39,800

●5″2DD、2HDの完全自動切替を実現しました。実際のVMと異なるのはメモリ容量(256KB)とクロック(8MHz)のみ、その他すべてVF仕様になります。8″バスにも対応。(ケーブル別売)
FDDの交換は不要、ショートプラグの変更のみで使用できます。

9801F2, F3, Eを2HD/2DD自動切替に改造.!!

8″内蔵2HD I/Fカード **PC-98M11mark II** キット価格 ¥89,000

●2HD専用、2DD専用、2HD/2DD自動切替いずれの方法もスイッチによる切替が可能です。●8″および2HD I/Fカードで最大2HD2台、8″が2台接続出来ます。または2HDが4台接続出来ます。●2DD切替機能が内蔵されています。●VFOカードが内蔵されています。●PC-9801Fにこのカードを使用し、内蔵フロッピーを2HD、2DD両用のフロッピーに取り替えることにより、2HD/2DDが自動切替になります。

NEC PC-9801用
高性能ROMライター **Rom Burner**

MR-01 本体価格¥38,500(ケーブル付)
＊電源を別途に必要

I/Fカード **PC-98M08** 定価¥7,200

●EP-ROM2716から27512までのすべてのROMの読み書き可●2764以上のROMは高速書込み可(27128で約50秒)●NEC PC-9800シリーズ全機種対応●電源はDC9V～12V用のアダプターを使用●I/FカードはPC98M08を使用●MS-DOS, CP/Mでコントロール, 専用ソフト付

＊MS-DOS, OS/2は米国マイクロソフト社、新松は㈱管理工学研究所、Macintoshはアップルコンピュータ社の商標です。

開発・販売元 株式会社 キャラベルデータシステム
〒150 東京都渋谷区渋谷4-3-17-606 ☎03(498)5370㈹
神戸出張所/〒651 神戸市中央区雲井通4-1-11ラ・エリール207☎078(261)8170

お求めは全国有名パソコンショップでどうぞ!

★商品価格には消費税は含まれておりません。

スタッフ募集中! 詳細はお問合せください

IBM AT互換 80386 33MHz

AST RESEARCH INC.

左から
AST Premium 386/33
AST Premium 386/25
AST Premium 386/C
AST Premium 386/16
AST Premium Workstation/386 SX

386/33MHz, 386/25MHzマシンについては486マシンへのアップグレードが可能です。

IBM互換機市場では、世界第2位の実績を誇る、ASTリサーチ社のプレミアム、コンピュータシリーズ。

そのラインナップに新たに、Premium 386/33MHzマシーンが加わりました。ソフトの開発、CAD/CAM、開発ツールとして既に米国市場では高い評価を受けた33MHzマシンを日本でもお使い頂けるようになりました。
また、多様化する日本のビジネス社会に対応するため、ASTリサーチ社では、386/25MHzマシンから286/8MHz(ゼロウエートステート)まで、多種多様な製品をとり揃え、あらゆるニーズにお応えすることができます。

	Premium 386/33 80386 33MHz	Premium 386/25 80386 25MHz	Premium 386/C 80386 20MHz	Premium 386/16 80386 16MHz	Premium 286 80286 10MHz (ゼロウエートステート)	Bravo/286 80286 8MHz (ゼロウエートステート)
RAM	: 2MB標準	2MB標準	1~2MB	1MB	512KB~1MB	512KB
最大メモリ	: 36MB	36MB	16MB	16MB	16MB	16MB
FDD	: 5.25インチ 1.2MB標準	5.25インチ 1.2MB標準	5.25インチ 1.2MB標準	5.25インチ 1.2MB標準	5.25インチ 1.2MB標準	5.25インチ 1.2MBオプション
HDD(MB)	: 110/320オプション	90/150/320オプション	40/90/150/320オプション	40MBオプション	20/40/70オプション	40MBオプション
拡張スロット	: 7	7	7	7	7	4
シリアル/パラレル	: 2/1	2/1	2/1	1/1	1/1	1/1
	キャッシュメモリ32KB搭載	キャッシュメモリ32KB搭載	キャッシュメモリ64KB搭載			
価格	: ¥1,300,000より	¥1,050,000より	¥750,000より	¥598,000より	¥428,000より	¥198,000より

※上記製品の他、Premium Workstation 286、Premium Workstation / 386SXもとり揃えております。 ※表示価格には消費税は含まれておりません

お問い合わせは：日本における代理店

スワイヤ トランステック 情報通信機器部
ジョン スワイヤ エンド サンズ(ジャパン)リミテッド 〒102 東京都千代田区一番町14スワイヤハウス
電話 03-230-9329(代表) FAX. 03-221-7957

PC ATはIBM社、i80286, i80386SX, i80386はIntel社の製品です。

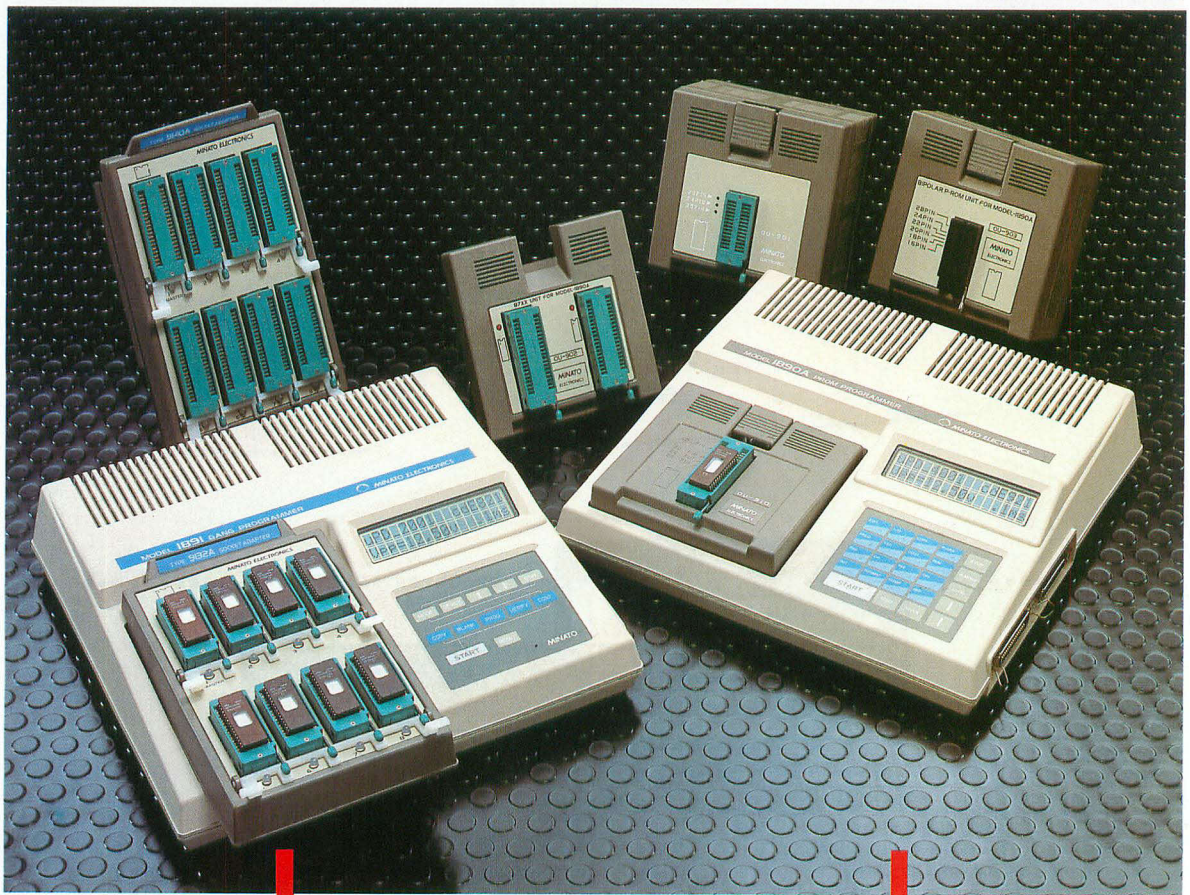
資料請求No. 20

MINATO

Next Measuring

4メガビット対応

この低価格で高速書き込み実現。



ギャングプログラマ MODEL 1891 ¥298,000

メガビット時代のコンパクトコピー専用ギャングプログラマ〈モデル1981〉は、薄型・軽量のボディにギャングソケットを搭載、標準で32ピン4Mビットまでの PROM8個にデータを同時にプログラミングできます。バッファメモリを装備しているのでマスターROMを選びません。視認性の優れた32文字ディスプレイは、PROMの型名やVpp電圧、書き込みパルス幅などはもちろん各種の設定項目をメニュー型式で表示するのでオペレーションは、快適です。

●16k～4MビットEPROM、EEPROM(32ピンまで)●シリコンシグネチュア機能●高速書き込み機能搭載●マスターROMを選ばないメモリ方式●200種以上のPROMをサポート●多彩な信頼性チェック機能●デバイスメーカ各社の書込みアルゴリズムを装備●4Mビットバッファメモリ装備

※40ピンメガビットPROM対応ギャングソケット〈TYPE9140A〉近日発売

書き込み時間データ(単位：sec.)

デバイス名	ブランクチェック	プログラム	ベリファイ	連続モード
P27512 (512Kビット)	1.5 sec.	14.5 sec.	5.5 sec.	21.5 sec.
i27270 (1Mビット)	3 sec.	22 sec.	10 sec.	35 sec.
μPD27C2001 (2Mビット)	5 sec.	43 sec.	20 sec.	68 sec.
μPD27C4001 (4Mビット)	10 sec.	87 sec.	39 sec.	135 sec.
TC574000 (4Mビット)	10 sec.	108 sec.	39 sec.	156 sec.

ユニバーサルプログラマ MODEL 1890A 本体 ¥128,000

ROMの進化を予見するミナトからユニバーサルプログラマ〈モデル1890A〉は最新のデバイスをサポート。MOS、PLD、のユニットがラインアップしました。こんなにコンパクトな低価格にもかかわらず拡張性やプログラミング性能は1ランク上の実力。使い易さを追求した本体機能は、それぞれのユニットの機能をフルに発揮します。

●RS232Cとセントロニクス準拠パラレルインターフェースを装備●2Mビットバッファメモリを装備。オプションで8Mビット拡張可●13種のデータフォーマットを装備●バイポーラユニットOU-903近日発売●87××ワンチップアダプターOU-902(オプション)

データが証明する高速書込能力

MOSユニットOU-910 ¥70,000

●書き込み時間(sec)

デバイス名	i27512 (512Kビット)	i27210 (1Mビット)	μPD27C2001 (2Mビット)
ブランクチェック	0.3	0.6	2.0
プログラム	12.0	11.7	41.0
ベリファイ	3.0	5.0	10.0
連続モード	15.3	17.3	53.0

●オリジナルLSIの採用で大量化に伴う高速書込を実現●4MビットまでのEPROM、EEPROMをサポート●豊富なエラーチェック機能●PROMのIDコードを自動判別

300種以上のPLDをサポート

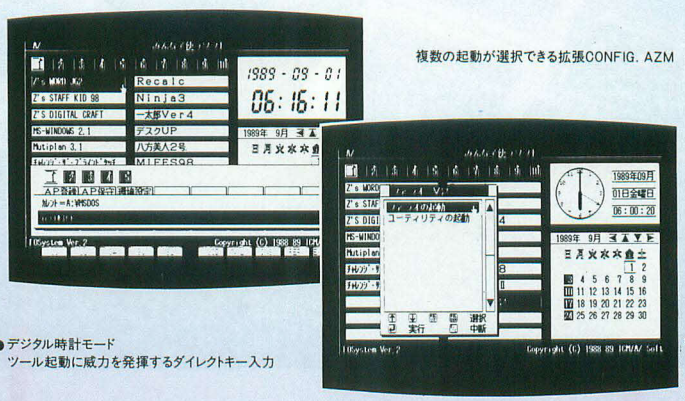
PLDユニットOU-901 ¥230,000

●デバイス選択はディスプレイで一目瞭然●大容量のテストベクタ領域を確保●PLD開発ツールからのデータ転送も万全

ミナトエレクトロニクス株式会社
本　社／223横浜市港北区南山田町4105　☎045(591)5611代
大阪営業所／530大阪市北区曽根崎新地2-6-21　☎06(348)9277代
北関東営業所／370高崎市双葉町6-25　☎0273(23)9701代
福岡営業所／812福岡市博多区博多駅東1-9-5　☎092(475)2825代
仙台営業所／980仙台市宮城野区福岡2-2-10　☎022(291)2571代

JEMIMA SHOW '89国際計測工業展 10月24日(火)～27日(金) 東京国際見本市会場 主催：社団法人日本電気計測器工業会 新館1F(N-29)

新しいEOシステムは もっとイージーオペレート



複数の起動が選択できる拡張CONFIG. AZM

デジタル時計モード
ツール起動に威力を発揮するダイレクトキー入力

EO System Ver.2

EOシステムVer.2の特徴(従来機能含む)
①市販ソフトが簡単にインストールできる。②1ページ20タイトル、最大200タイトルのアプリケーションがインストールできる。③インストールした市販ソフトがメニュー形式で簡単に起動できる。④日本語FEPがリセットすることなく、切り換えることができる。

EOシステムVer.2の特徴(Ver.1追加機能)
①ツール起動に威力を発揮するダイレクトキー入力機能②複数の起動が選択できる拡張CONFIG. AZM③CONFIG. AZMなどを変更するミニエディタ搭載④印刷機能付スクリーンエディタ搭載⑤簡易デバイスドライバ登録機能⑥サブメニューはコマンド直接起動型でツール起動に最適⑦メニュー画面タイトル設定機能⑧メモリースイッチのブート装置の変更設定機能⑨ソフトウェアリセット機能⑩強力なファイル管理機能⑪柔軟な印刷機能⑫液晶画面には欠かせない画面反転モード⑬データ用FDフォーマット機能など。

※プロテクトのかかっている市販ソフトに関してはインストール手順の異なる場合もあります。
※IBFファイルは㈱アイシーエムと㈱エージーソフトが開発したインストール手順ファイルです。
※会社名、商品名は各社の商標、または登録商標です。

●以下の商品は標準出荷時にIBFファイル付で出荷されています。

▶ねじ式・ジーズトリフォニデジタルクラフト・ジーズワードJG Ver.2/株式会社ツァイト▶Success/株式会社サムシンググッド▶Recal/株式会社アスキー

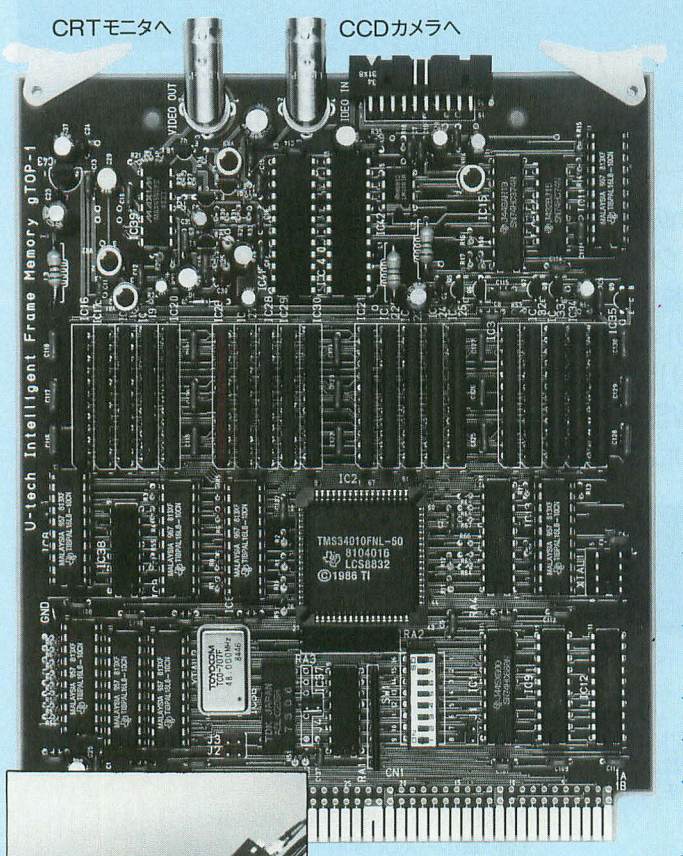
■STRIDEシリーズ●SR-20/¥98,000●SR-20(L)/¥98,000●SR-40/¥118,000●SR-40(L)/¥118,000●SR-60/¥148,000●SR-60(L)/¥148,000●SR-80/¥198,000●SR-80(L)/¥198,000 ■LESAGEシリーズ●HD-6040HS/¥290,000●HD-6060HS/¥350,000●HD-6155ES/¥490,000

data show '89 アイシーエムは、『人と情報環境の新しい調和』をテーマとした、データショウ'89に出展いたします。(西館W-3)

ICM 株式会社アイシーエム
●本社営業部/〒556 大阪市浪速区日本橋4丁目2番9号 TEL. 06(648)4702 FAX. 06(647)2018
●東京営業所/〒101 東京都千代田区神田小川町2丁目2番 サンブッデビル2F TEL. 03(291)7491 FAX. 03(291)7494
●名古屋営業所/〒450 名古屋市中村区名駅南2丁目14番19号 住友生命名古屋ビル10F TEL. 052(561)4927 FAX. 052(561)3644
●北海道営業所/〒060 札幌市中央区北1条西2丁目1番地 時計台ビル11F TEL. 011(210)4778 FAX. 011(210)4766
●開発スタッフ募集中/お気軽にお問い合わせ下さい(総務部06-647-2008まで)

インテリジェント フレームメモリ

model gTOP-1



gTOP-1は画像処理プロセッサを搭載したフレームメモリボードでPC98バスに直接スロットインでき、CCDカメラから画像を取り込んで寸法計測、形状検査等を高速かつリアルタイムにオンボードで処理できます。

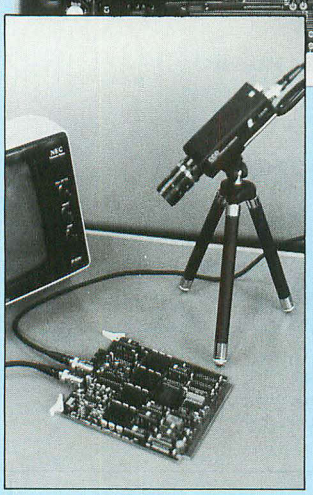
- ★最大1024×512 プログラマブルピクセル
- ★モノクロ256諧調コンポジットビデオ
- ★32 bit GSP TMS34010プロセッサ
  - ●強力なグラフィック命令
  - ●32 bit×30本 アキュムレータ
  - ●256バイト命令キャッシュ
  - ●160nsec CPUサイクル
- ★512KBビデオRAM+128KBのプログラム/DATAメモリ
- ★X-Yアドレッシング及びI/Oポート R/W使用により面倒なセグメント管理不要。
- ★CCDカメラ用同期信号、ストロボ同期出力。
- ★並列処理中も表示画面のチラツキなし。
- ★画像取込後即処理できるのでパソコンへの大量データ転送不要。
- ★GSPのプログラミングをせず、PC98から直接ビデオRAMへのR/Wも可能※

〔※この場合にはオプションソフトは不要です。〕

- ★マルチカメラシステムも簡単に構築可。
- ★オプションソフト
  - ●GSP "C"/アセンブラ/リンカ/シミュレータ（TI製）
  - ●デバッガユーティリティ（ユーテック製）

このボードだけで経済的なGSPプログラムの開発環境が出来上ります。

¥198,000



⇧
ソニーXC-77CE
接続例

■主要商品■レーザー・スキャン・アナライザ、超音波データ解析システム、電子ビーム描画制御、超高速演算回路、画像処理システム、プロトコルモニタ PROTO-Ⅲ

Utech 株式会社 ユーテック

〒532 大阪市淀川区西中島3丁目10番13号
新大阪物産ビル10F
TEL 06(306)4641 FAX 06(301)4628

最高のファームウェア開発環境を提供します

μ-Series

ホスト環境：VAX、Sun、HP-9000、MS-DOS

μ-SeriesはSwedenのIAR社で開発されたROM化のための
クロスソフトウェアツールで、クロスCコンパイラ、クロスアセンブラ、リンカ、
シミュレータなどのファミリにより統合的なファームウェア開発環境をサポートします。
μ-Cおよびμ-ASMがサポートするターゲットCPUはシングルチップから32ビットまで18種類を数え、
関連チップを含めると約110種のCPUプログラムに対応できます。
また、ホスト環境も各種EWSからMS-DOSまでをサポートしており、
ユーザーセレクタブルなマルチホスト／マルチターゲット開発環境を提供します。

μ-C
クロスCコンパイラ

■ROM化をサポート
ROM化をターゲットとしたコンパイラで、Cの全ステートメント、全データタイプ、型定義、long、float、初期化されたstatic変数などをサポートします。また、ROM化用のCのスタートアップルーチンがアセンブラプログラムにて提供されます。

■ANSI規格に準拠
ANSI規格に準拠(K&Rの規格を含む)し、各マイクロプロセッサ用の機能が付加されています。
- const、volatile、signed、void、enumキーワードの追加
- 関数プロトタイピングの導入
- struct、unionの割付範囲の拡大
- #elif、#errorなどのプリプロセッサ疑似命令の追加

■オプティマイズコンパイラ
コンパイル時に以下のような最適化を自動的に行います。
- 最短なジャンプ／分岐命令の使用
- 不用なジャンプ命令の削除
- 乗算をシフト演算にて代行
- 単精度浮動小数点演算
- スイッチ文への最適なアルゴリズムの適用(ジャンプ、スキャン、ソートテーブル)
- ゼロページオブジェクトのサポート

■ハイスピードコンパイラ
主記憶をベースとしたワンパスのコンパイラで以下のようなアーキテクチャにより高速化が計られています。
- テンポラリファイルを作らない
- アセンブリパスがなく直接オブジェクトを出力する(スイッチによってアセンブラコードも出力可能)
- プリプロセッサのパスが分離されていない

■豊富なライブラリ関数をサポート
全CPU共通に41種類のCライブラリ関数が用意されており、この他、各CPU毎にインライン関数、拡張I／O関数など独自の関数がサポートされています。

■パワフルな型チェック機能
UNIXのLINT並の型チェック機能が組み込まれており、パスカルライクな警告メッセージを出力します(-gオプションを使用)。

■複数のメモリモデルをサポート
CPU毎に複数のメモリモデルが用意されており、マイクロコントローラのメモリ仕様に合わせて選択することができます(-mオプションを使用)。また、メモリのバンク切換えもサポートしており、256個までのメモリブロックを使用できます。

■printfモジュール
printf／sprintfの構成モジュールが目的別に3種類用意されています。
- ANSI
- ANSI(浮動小数点を除く)
- 縮小版(ソースにて提供)

■コンパイラスイッチ
主なオプションスイッチ：
- -g：グローバルタイプチェックを有効
- -m：メモリモデルの指定
- -s：スピードの最適化
- -z：サイズの最適化
- -j：スタティック、ローカルシンボルの生成
  (同時にソース行番号情報を生成)
- -e：CPU別拡張関数を有効(インライン関数など)
- -l：リストファイルの生成
- -x：クロスリファレンスの生成
- -a：アセンブラニーモニックをリストファイル上に挿入
- -a:アセンブラソースの生成

ベンチマーク　その①

●コンパイル時間(単位：秒)

	/50	/60	/260
μ-C	21.0	13.0	9.2
H社	167.0	99.8	75.5
L社	346.7	209.2	152.2

※ホストはSun-3/50、/60、/260を使用
※対象プログラムはANSI準拠のソースで、オブジェクトサイズは19KByte

μ-ASM

クロスマクロアセンブラ

基本命令は、各チップメーカー供給のアセンブラと高い互換性を有しているため、命令レベルでは今までのソフトウェア資産を有効に利用することが出来ます。
各チップメーカーのアセンブラと互換性のある言語仕様部分：

- マシン命令　・ラベル　・定数定義疑似命令
- 定数　・エリア確保疑似命令　・基本演算子(＋、－、＊、／)
- デリミッタ　・ORG、EQU

■主な機能

- INCLUDEの深さは3レベルまで可能
- リロケータブル・セグメントの制限なし
- 32ビット演算および単精度浮動小数点定数をサポート
- シンボル名は255文字まで使用可能
- 27種類の演算子をサポート
- リカーシブ可能なマクロ機能により強力なマクロ処理を実現
- コンディショナルアセンブルのネスト制限なし
- クロスリファレンスとリスト出力のコントロール可能
- オブジェクトモジュールにローカル／グローバルシンボル出力可能

■アセンブラスイッチ
主なオプションスイッチ：

X　クロスリファレンス、シンボルテーブルの生成
P=nn　ページ指定(行／ページ)
E　エラー行のみをリストファイルとして生成
S　ローカルシンボルをオブジェクトファイルに出力
?　オプションメニューの表示

ベンチマーク　その②

●アセンブラ実行時間(単位：秒)

	実行時間
μ-ASM	9
I 社	32

※ホストはPC-9801V,M(8MHz)、MS-DOS
※対象プログラムサイズ：約400 Byte

μ-SIM

シミュレータ

■基本機能
μ-SIMの基本機能は以下の通りです。

- 完全なインストラクションセットでのシミュレーション
- シンボルテーブルのハンドリング
- シミュレーションメモリ内容の変更
- インストラクション形式又は16進形式でのメモリ内容の表示
- ワンラインアセンブラによる実行インストラクションの変更
- 実行時間(マシンサイクル)の計算
- 変数およびレジスタの変更
- インストラクションまたはデータのR／Wに対する条件ブレークポイント
- シングルステップによるシミュレーション
- I／Oシミュレーション(会話的に、もしくはファイルへ)
- 割り込み処理のシミュレーション
- メモリ構成(ROM／RAM)の管理
- ユーザ定義関数のサポート(マクロ機能)
- ログファイル(全コマンドの記録)
- システムコマンドの実行

μ-LINK/LIB

ユニバーサルリンカ/ライブラリアン

■特長
μ-Seriesの専用ユニバーサルリンカで全てのμ-C、μ-ASMに共通に使用できます。

■出力形式
出力ファイルはIntel、Motorola、Millenium、HPなど36種類のASCIIまたはバイナリフォーマットから選択可能で(-Fスイッチを使用)、主要ICEメーカの製品にてCのソースレベルまたはシンボリックでのデバッグが可能です。

■リンカスイッチ
主なオプションスイッチ：

-I：リストファイルの生成
-X：クロスリファレンスの生成
　・プログラムエントリのアドレスとモジュール名
　・モジュールマップ　・セグメントマップ
　・インデックスリスト
-pnn：ページ指定(行／ページ)　-C：CPUの指定
-F：出力フォーマットの指定　-f：コマンドファイルを有効
-b：バンクスイッチングを有効
-DSYMB：コマンドラインからのシンボル定義
-Z：セグメントおよびセグメントアドレスの指定

■バンク割付
リンク時にセグメントのバンクへの割付指定が可能です(-bオプションを使用)。

start → Main memory
start+len → First bank
start+inc → Second bank
start+inc+len → Second bank
start+2*inc → Third bank
start+2*inc+len → Third bank

■ライブラリアン
μ-LINKパッケージにはライブラリアンとしてμ-LIBが含まれており、次のような機能を有しています。

- オブジェクトコードの追加、削除
- コマンドストラクチャとエラーメッセージ等の情報出力
- オブジェクトモジュールに対するリスト、デリート、ムーブ、アップデート、リネーム、アペンド等

■μ-Series製品一覧表　▲印は開発中です

	μ-C	μ-ASM	μ-SIM		μ-C	μ-ASM	μ-SIM
804X		●		Z80	●	●	●
8051	●	●		64180	●	●	
8085		●	●	TMS7000		●	
8096	●	●		TMS9900		●	
6801/6301	●	●	●	1802		●	
6805		●	●	6502		●	
6809		●	●	7800	●	●	
68HC11	●	●	●	H16	▲	▲	
68000/10	▲	●	●	H8/500	▲	▲	
Z8		●		H8/300	▲	▲	

■μ-Series価格表　※表示価格には、消費税は含まれていません。

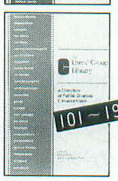
	μ-C	μ-ASM	μ-SIM	μ-LINK
VAX全機種	¥600,000～	¥400,000～	¥500,000～	¥400,000～
Sun HP-9000	¥600,000	¥400,000	¥500,000	¥400,000
MS-DOS	¥148,000	¥ 98,000	¥148,000	¥ 98,000

株式会社 LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE：03-293-4711 FAX：03-293-4710
TELEX：242-3296

お問い合せ
営業1課
TEL 03-293-4714

PDS/Shareware Best Sallers

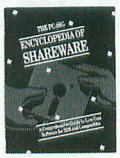
CUG (The C Users' Group)



1. ＃285 BISON
2. ＃198 MicroEmacs V3.9 Source
3. ＃271 Steven's Library(Turbo €)
4. ＃258 C Tutor Source (Turbo C)
5. ＃155 B-TREES. FFT. etc
6. ＃263 C_wndw Toolkit
7. ＃197 MicroEmacs V3.9 Exec. & Doc.
8. ＃163 Small C for MS-DOS
9. ＃289 Othello
10. ＃231 Little Smalltalk (UN-Partl)
11. ＃232 Little Smalltalk (UN-Part 2)
12. ＃255 EGA Graphics Library
13. ＃172 LEX partl
14. ＃173 LEX part2
15. ＃257 C Tutor Doc (Turbo C)

Minix & Fixes ¥3,000 〒1,000
1Vol ¥1,500 〒175
Library(book) 101〜199 ¥2,000 〒260
New Library(book) 200〜284 ¥2,000 〒260

PC-SIG (IBM-PC PC-DOS)



1. ＃1080, ＃1081 Modula-2 Compiler(2disk set)
2. ＃148 X-LISP V1.7
3. ＃1120, ＃1121, ＃1122 Black Magic(3disk set)
4. ＃980 Vanilla Snobol 4 Programmers Utilities
5. ＃685 NewFIG FORTH V1.0
6. ＃433, ＃434 Kermit-MS
7. ＃147, SDB(Simple Datebase)
8. ＃928 Snobolyt Utilities V1.0
9. ＃965 Mystic pascal
10. ＃801 Sail

1vol ¥2,000〒175
5thLibrary¥3,800〒310
CD-ROM(ALL) ¥69,000 〒1,000

The C Users Journal
日本語版 ¥1,800(〒0)

- The C Users Journal 米国C言語専門誌、年12冊 ¥1,000 〒260 1年購読 ¥12,000(〒込)
- PROGRAMMER'S Journal ¥1,200 〒210
- DDJ ¥1,000 〒260
- COMPUTER LANGUAGE ¥1,000 〒260
- THE IBM PC&PS/2 ¥4,650 〒370

Best Prices SOFT WARE

ACT.SYS+IBMPC.OPT Ⓜ¥9,000〒¥★
Turbo ASSEMBLER & DEBUGGER Ⓜ ¥31,000 〒¥★
Turbo C 2.0(MS-DOS)¥23,000 〒¥★
Top Speed Modula-2 Ⓟ DOS-3 PACK ¥40,000 〒¥★
dbXL(PC-DOS)¥39,800(MS-DOS)¥49,800〒 ¥★
FTL MODULA-2(MS-DOS)¥19,800 〒¥★
SmallTalkV(MS-DOS)¥49,800(PC-DOS)¥39,800〒¥★
Small Talk V Goodies ＃1 (MS-DOS)¥16,800 〒 ¥★
OPTASM Ver.1.6 (MS-DOS) ¥16,000 〒 ¥★
Sourcer Ⓟ **¥19,800** 〒¥★
JUG-PDSカタログディスク(MS-DOS 5"2DD)¥1000 〒0

Sourcer+Bios P.P (PC-DOS) ¥32,000 〒¥★
Turbo Prolog 2.0(PC-DOS)¥27,000 〒¥★
Eureka the Solver ¥30,000 〒¥★
OPTLINK Ⓜ **¥16,800** 〒¥★
DSD87 (MS-DOS/PC-DOS) ¥23,800 〒 ¥ ★
SmallTalk V 286 ¥69,800 〒¥★
SmallTalk V Goodies ＃2, ＃3 (PC-DOS)各¥16,800〒¥★
QUICKSILVER (MS-DOS) ¥165,000 〒 ¥★
OPTLIB (MS-DOS) ¥9,800 〒¥★
UTAH COBOL (MS-DOS) ¥14,800 〒¥★
UTAH FORTRAN (MS-DOS) ¥9,800 〒¥★
Turbo Pascal 5.0 (MSA) ¥23,000 〒¥★
Turbo Pascal 5.0 Professional Ⓜ **¥45,000** 〒¥★
Turbo C 2.0 Professional Ⓜ **¥45,000** 〒¥★
Turbo Pascal Toolbox Numerical Methods Ⓟ ver1.0 ¥5,500 〒¥★
T-Debug Plus Ⓟ ver 2.0 ¥3,000 〒¥★
Turbo Editor Toolbox Ⓟ ver 1.0 ¥7,000 〒¥★
Turbo Prolog Toolbox Ⓟ ver 1.0 ¥5,500 〒¥★

·PHARLAP SOFTWARE DEBUG
·NOVELL B-TRIEVE
·FASTBACK PLUS 2.0
·GRAMMATIK III
·MATH BLASTER PLUS
·MICROSOFT FLIGHT SIM.
·MICROSOFT MACRO ASSEMB.
·MICROSOFT QUICK BASIC
·MICROSOFT QUICK C COMPILER
·MICROSOFT WINDOWS 286
·MICROSOFT WORKS
·PC TOOLS DELUXE 5.5
·READER RABBIT
·SIDEWAYS 3.2
·TETRIS
·TIME LINE
·VM386
·WORDPERFECT 5.0
·BORLAND QUATTRO
·CARBON COPY PLUS 5.0
·HARVARD GRAPHICS 2.12
·NORTON UTIL.ADVNCD 4.5
·PAGEMAKER 3.0
·PFS:FIRST PUBLISHER
·BORLAND TURBO C 2.0
·BORLAND TURBO PASCAL 5.5
·VP-PLANNER+
·CROSSTALK XVI
·TYPING TUTOR IV

CALL TODAY

ALL IBM FORMAT

提供 ソフトプロダクトセンター 東洋エンジニアリング株式会社

MS Windows' world

★Guide
★Click Art
★PC Paintbrush for Windows
★Micro grafx Designer
★Hypertext-Based Pocument Manager

CALL TODAY

送料★：¥1,000
Ⓜ＝MS-DOS Ⓟ＝PC-DOS

dBXL 98専用版
¥49,800→¥39,000〒★

MINIX BINARIES
AND SOURCES
for 640K IBM PC'S
version 1.3
(PC-DOS)
¥18,000 〒1,000

ZASM for Z80&HD64180 **¥10,000**
MS-DOS上で動く2パスの環境型ソロケータブル・アセンブラーです。ソースはTURBO-C(V2.0 TC.EXE)でコンパインできオールCです。

ZZ for Z80&HD64180 **¥10,000**
MS-DOS上で動く、スクリーン・エディター・タイプの逆アセンブラーで、全ソース・ファイル付きです(TURBO-C2.0TX.EXEでコンパイル)。

98用通信ソフト
Picoterm
¥19,000 〒1000
●話題のMNPクラス6をサポート ●超高速パック・スクロール ●操作性抜群

Software from VAINS

dBASE Ⅲ Plus対応
dLIST ¥15,000 〒1,000
(98,FMR,IBM PS/55)
構造化プログラム言語であるdBASEの構造のティバッグとプログラム、資料、作成に重点を置いた開発サポートツールです。

dGRAPHICS(98) ¥15,000 〒1,000
グラフィクス機能やウィンドウ機能を可能にするアプリケーションです。

dPRINTER(98) ¥10,000〒1,000
dBASEⅢ Plusで罫線印刷機能を可能にするアプリケーションです。

dRS-232C(98) ¥15,000 〒1,000
今までdBASEでは考えられなかったRS-232Cのコントロールを可能にしたアプリケーションです。計測器からのデータの取り込みやプロッターへの作図が可能です。

最新VC-PDS @3,000 〒175

＃13 PROFILER & MS-DOS TOOLS by ksoq
プロファイラを初めとした7本のツール9801/286
＃14 スクリーンエディタ F(えふ) by FENIX
FM11用(MS-DOS2.11以上, メモリ384K以上, 文節変換vje.sysが必要)
＃15 スクリーンエディタ JE by はじめ
J31 Turbo Pascal ソース入り
＃16 名刺管理 Name by cherry
9801/286用の名刺管理 Data Base
＃17 Power C Graphics Interface (PCGI) by Crosswell
9801/286用ソース別売1,500円
＃18 MicroEmacs ver 3.9 MS-DOS インストールプログラム by YAMA (1,500円)
初心者向け CUG 198 (MicroEmacs ver 3.9 Sourse)をMS-DOS版にコンパイルするためのユーティリティ。
対応：Turbo C 1.5, Lattice C 3.1, Dateright C
＃19 MS-DOS文書処理用ユーティリティ by Toshi
アフリカ マラウィにて現地語、英語、日本語の対訳テキストを小人数で編集する必要から作成したプログラム。
＃20 Vz editor 1.01対応 FM版 (1,500円) by K. Tok
対応機種：FM16β, FMR50/60/70 FMTOWNS(MS-DOS) 98用Vz購入者のみの配布となります。

Vz Editor 定価 9,800円

対応
NEC 9800シリーズ
EPSON 286シリーズ
メディア：5"2HD、3.5"2HD

Ver1.5 '89.10月末発売予定
PC9801、PC286、J3100、IBM PC、AX、EMSフルサポート予定
サンプル版配布開始(1000円)

文書ファイルを数多く使う人のための超高速軽量エディタです。

●論理ページング機能● 'R,'Cで、/*..*/;====等で始まるタイトル行をサーチする、論理ページングモードが使えます。●ブロック境界●ブロックモードでは、カーソルの移動方向により、行単位／文字単位の境界が自動的に切り替わります。もちろん矩形ブロック処理も可能です。●日本語処理●各種FEPを自動的に識別し、ON／OFFを制御します。また、'F,'Aのワード単位の移動は、日本語に対応しています。●アペンドコマンド●指定ブロックを既存のファイルの後ろに追加するアペンドコマンドは、通信、ログファイルの編集に欠かせません。●大きなファイルの編集●十分なディスクの空きがあれば、編集可能なファイルサイズに制限はありません。また、編集テキストのリード／ライトは、copyコマンドと同じスピードです。●DOS環境ツール●ヒストリー、エイリアス、DOS画面出力の取り込み機能を装備しており、VZを常駐させれば、コマンドラインからいつでもこれらの機能が使えます。●ファイラー●２つのディレクトリを同時表示し、複数ファイルの編集・コピー・削除等を、ファイルを選択するだけで実行できます。●ユーザーカスタマイズ●キー配置、メニュー、タイトル、カラー等は、定義ファイルを書き換えるだけで自由に設定できます。mifesライクな定義ファイルも付属しています。●マクロ言語●VZのマクロ言語は、次のような特徴を備えています。★39種のCライクな演算子(push, pop, byte/word ptr演算あり)。★56個の汎用変数の他、オプション変数、システム変数を演算式で使用可能。★24種のシステム関数（printf, int86, strchr, locate, rand,相当の関数あり)。★条件分岐、多重分岐、ブロック、ループ制御構造。★マクロの最大８レベルのネストおよびバックグラウンドでの実行。★編集中のマクロプログラムをその場で組み込み、実行可能。★トレース時に、実行箇所のマクロプログラムを画面に表示。★任意の文字列をマクロプログラムとしてコール可能。

オールアセンブラフルスクリーンエディタ
超高速・高機能・低価格・ソースリスト付
動作OS：NEC製MS-DOS 2.11以降
動作メモリ：256KB以上
(他のソフト組み込みによる)
ディスプレイ：漢字表示可能なもの

VILLAGE CENTER
〒101 千代田区神田神保町2-2-34 千代田三信ビル7F
米国The C Users Group日本支部 TEL 03-239-1093

■通信販売ご希望の方は 〒 振替か、現金書留にてお申し込み下さい。
東京 9-350878 口座名 ビレッジセンター
価格には消費税は含まれておりません。

平日 13:00〜18:00
土曜 11:00〜19:00
日・祭休

SAVE MONEY! SAVE TIME!
DON'T WAIT ORDER NOW!

The C Users Journal Japan Publisher Village Center Inc.

創刊2号10月18日発売

1800円(送料・税込み)

創刊2号目次予定

豪華新連載3本決定!アメリカからお届けします。

♠夢のまた夢 ＮｅＸＴ:吉田弘一郎
♠計算機屋のためのアメリカ経済解説:ﾙｲｼﾞ熊山
♠OS/2ﾃﾞﾊﾞﾛｯﾊﾟｰ･･･ハーゴのつぶやき:ﾊｰﾊﾞｰﾄ ｼﾞｮｰﾝｽﾞ
☆Standard C プリプロセッサ
☆Doctor C's Pointers いくつかの場面では typedefは#defineよりも優れている
☆高速ｳｫﾙｼｭ変換
☆C及びC++のためのスキャフォルディング
☆GEMディスプレイのCプログラムの標準的方法
☆Vitamin C 総合的な画面制御ライブラリ
☆Amigaユーザのための本物のlint
☆Cソース中へのSQLコマンドの組み方
☆Turbo Cによるデバイスドライバの作成
☆DSPをCでプログラムする
☆ハードウェアによる滑らかなスクロールとパン
☆AppleIIgs上の簡単なテキスト出力
☆Questions & Answers ISAMとｳｨﾝﾄﾞｳのﾗｲﾌﾞﾗﾘ構築
☆UNIX Tools
☆神麦人:独断偏見読書録②

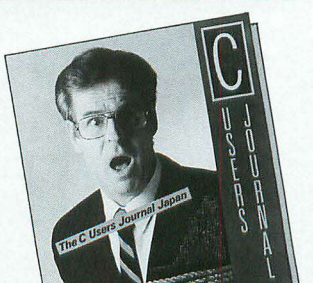
創刊号目次

★Standard C 関数の宣言
★Cの移植性の指針に関する発行物
★設計目標としての移植性
★オーバーレイ用のUNIXの代替品
★SHERLOCK もうひとつのデバッガ
★Zortech C++
★Dr. C's Pointers sizeofの微細な側面を探る
★UNIX・MS-DOS間のアプリケーション
★移植性の為の設計とコーディング上の検討
★戦術をプログラムしてプレーするゲームCROBOTS
★リビジョン・コントロール・システムズ
★fscanf()を用いた構文解析プログラム
★MS-DOS PCのタイミング分解能の改善
★強力なデータカプセル化によってリングバッファを構築する
★連結リストのソートのための効果的アルゴリズム
★MS-DOSのデバイスドライバをCで記述する方法
★QNX 環境上におけるタスク間通信
★UNIXのソースを89ドルで買いたいと思いませんか
★ロジックを簡単化するツール
★神麦人:独断偏見読書録①

創刊号完売間近!

お申し込みはお早めに

2号より新連載の「夢のまた夢 NeXT」では、吉田弘一郎氏がサンフランシスコよりNeXTをあらゆる角度から実況生中継致します。



申込方法

1. 直接購読
   郵便振替 口座番号東京9-350878 口座名(株)ビレッジセンター
2. 書店購読
   お近くの書店に(株)技術評論社 発売 (株)ビレッジセンター 発行 The C Users Journal Japan とお申し込みください。

株式会社ビレッジセンター
101 東京都千代田区神田神保町2－2－34
千代田三信ビル7F 03(239)1950

資料請求No.27

Telecomet Japan Inc.

ビューティフルに。
よりスピーディに。

MULTIFAXは、PC-9801シリーズのパソコンに内蔵して、
高速同期データ通信を行う汎用通信アダプターです。

パソコンでFAX
MULTIFAX

特　長

- 送受信原稿の内容をリアルタイムに画面表示。
- FDからの直接送信、受信内容の自動ファイル機能(MS-DOS上)。
- イメージスキャナやFAX装置からのイメージ原稿の取り込み
- 時間指定送信、同報送信も可能。
- 24ドット、32ドット高品位フォントを利用した送信。
- 独立したRS-232Cシリアル I/Oポート
- T-modemプロトコルによるエラーフリー通信ソフトウェア(ファイル転送機能)付。
  ファイル転送する場合、イメージ情報に変換せずに、キャラクタファイルとして高速転送しますので受信した情報は文字情報として受信側で加工することが可能です。所用時間は従来の1200bpsXモデムプロトコルの$\frac{1}{10}$以下の通信時間でエラーフリー通信ができます。

●主な仕様

装置型式	送信/受信兼用型ファクシミリボード
通信方式	G3ファクシミリ
送信原稿サイズ	最大A3
受信原稿サイズ	最大A3
通信速度	4800/2400BPS
符号化方式	一次元MH方式
走査線密度	ノーマルモード(3.85ℓ/mm) ファインモード(7.7ℓ/mm)
使用回線	一般加入電話回線
使用可能機種	PC-9801シリーズ(XA/LTを除く)
必要メモリ容量	384KB以上
ダイヤル方式	PB/DP(20/10PPS)(自動判別機能付)
通信可能相手機種	G3規格のファクシミリ全機種
使用電源	パソコンの拡張スロットより供給
外形寸法	170×150×22mm

(32ドット明朝体印字サンプル)

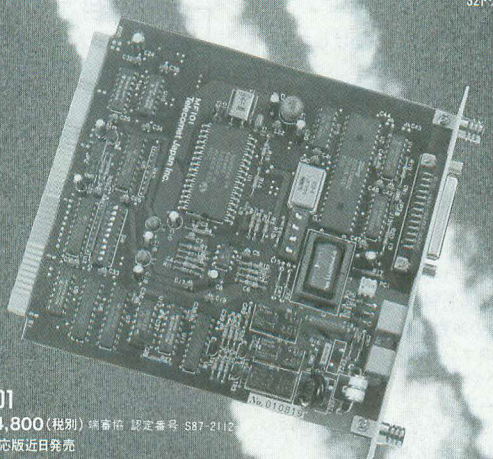
●マルチファックスからFAXへの送信

通信可能相手機種
G3規格ファクシミリ全機種

●FAXからFAXへの送信

通信可能相手機種
G3規格ファクシミリ全機種

32ドットはオプションです。



MF-101
価格 ¥64,800(税別) 端番符 認定番号 S87-2112
9600bps対応版近日発売

★ソフトのオンラインメンテナンス	日本テレコメット㈱では、サポートセンターを常設。マルチファックスのデータ通信機能を使用して、お客様に直接ボードの機能チェック、ソフトウェアのバージョンアップや使用方法等をオンラインで提供致します。
★ソフトウェア開発会社の皆様へ	特定業務用アプリケーションを開発しようとするシステムハウス様向けには、他のソフトからマルチファックスを制御するのに必要なハード及びソフト情報の提供もいたしております。また、JUST-PCやX.32に対応することも可能です。ご相談下さい。

日本テレコメット株式会社　〒163 東京都新宿区西新宿2丁目7-1 新宿第一生命ビル
新宿第一生命ビル内郵便局 私書箱5044　Tel:(03)343-1806 FAX:(03)348-8070 サポートセンター Tel:(03)343-632

Bの世界戦略

欧米でトップクラスのエディタ「Brief」の国際感性をあなたに!!

■ハイレベルな要求に応える高機能

●豊富なエディット機能

自動インデント機能／テンプレート機能／ワンタッチ・コンパイル機能／エラー・ジャンプ機能／タグジャンプ機能／検索・置換機能

●確かな安全機能

マルチ・アンドウ機能（最大300）／オートセーブ機能

●その他、多彩な機能を装備し、初心者から実務家まで、あらゆるレベルのプログラマに対応しています。

■マシンを問わない多機種対応設計

Briefはドライバーと呼ばれるソフト部品を取り替えるだけで、全く別のメーカのマシンでも利用できる様に設計しています。そのためパッケージには複数のドライバーを含めています。また、5インチと3.5インチの両方のメディア（2DD）を一括提供していますので、お手持ちの複数の機種でご利用いただけます。この経済性と共通の操作性を持ったBriefは、今までパソコンソフトでは実現できなかった目標をクリヤーしたといえましょう。

■実行環境

●対応機種：AXシリーズ、J-3100シリーズ、PS/55 T/Sシリーズ、5550MPシリーズ、PC-9801シリーズ

●対応OS：MS-DOS 3.1以上

●必要メモリサイズ：384KB以上

●最小ファイルサイズ：ハードディスク720KB以上、フロッピーディスク640KB以上

●ウインドウ数：最大55

●同時処理可能ファイル数：特に制限なし

定価 48,000円
（消費税は含まず）

日本システム株式会社 立川事業所
Briefユーザー・サポート係

〒190 東京都立川市曙町1-18-2 一清ビル TEL.0425-23-4469 FAX.0425-27-6127

MS-DOS・PC-9801(ハイレゾ含む)専用 手抜き作業はいけません!!

Microsoft-C 3.0以上
Lattice-C 3.0以上
MS-FORTRAN 4.0以上

ソフト開発担当様

請負受託ソフト開発でscanf READやprintf WRITE命令でお客様の満足いくプログラムが開発できますか？PC-9801で実行するプログラムでは、キーボード・データ入力中にINSやDEL、ファンクションキーなどの制御編集キーをサポートするのがあたりまえ。ＢＳキーしかサポートしないプログラムを無理矢理納品したら、開発技術力を疑われます。お客様の立場になって『使いやすく満足いく』プログラムを開発していない（開発できない？）、いわば「手抜き作業」をしている人々に当社では以下のライブラリを提供します。

1. モニタ画面表示処理の悩みを解決する
『テキスト画面高速描画処理ライブラリ』

マニュアルのみ 2,400円
ライブラリのみ 12,800円
ソース付 25,000円

a）テキスト画面表示処理関数73個、付属処理関数18個の91個の関数収録。大部分の関数はアセンブラ言語で記述していますので、初代98でも画面表示が信じられない程高速です。本製品はサービス商品のため格安ですが、製品内容は価格に似合わず充実しています。

b）裏画面やメモリ画面をサポート。ユーザの用意した６ＫＢ配列領域をテキスト画面と同様に処理する機能あり（メモリ画面）。この機能により、メニュー画面を何枚も表示するようなプログラム開発が容易にできます。

c）オリジナル・テキストウィンドウの作成をサポート。テキスト画面指定領域の文字データを退避したり復活したりする関数や、縦横スクロール処理などをアセンブラで作成した関数を提供します。

2. キーボード・データ入力処理の悩みを解決する
『キーボード・データ入力処理ライブラリ』

マニュアルのみ 2,400円
ライブラリのみ 28,000円
ソース付 58,000円

注）本製品を使用する場合は「テキスト画面高速描画処理ライブラリ」が必要になります。

a）キーボード・データ入力処理関数77個、付属処理関数42個の119個の関数収録。ＲＸ/ＲＡの拡張キーボードやＦ11～Ｆ15の拡張ファンクションキーに対応。

b）整数入力、固定小数点入力、e形式入力、文字列入力、メニュー選択入力など、プログラム開発に必要な基本データ入力処理を全てサポート。数値系入力ではBASIC・INPUT命令型（フリーフォーマット）と、整数・小数・指数部桁数を固定した入力をサポート。用途に応じて使い分けます。
指定されたフィールド内で文字列入力を行う関数や、本製品の"目玉関数"である「横スクロール形式文字列入力関数」をサポート。この関数は、ユーザーの指定したフィールド内に、指定されたバイト数を入力する関数です。20カラムのフィールドに漢字を100文字入力したい時の要求を解決する関数です。なおこの関数は、Ｃ言語で1000ステップを越える大きな関数で、3000カラム5000バイト程度の文字列入力が可能です。

C）お客様の立場に立って、データ入力中ではINS、DEL等のPC-9801の制御キーを全てサポート。もちろん制御キーの有効・無効の指示ができます。

〔本ライブラリの開発方針〕

a）関数を呼び出すだけで、市販ソフト並の画面表示やデータ入力が実現できること。
b）ノーマルモード98とハイレゾモード98の動作モードを識別し、どちらの機種でも実行できること。
c）98のハードウエア能力を最大限生かすライブラリであること。従って本製品を使用したプログラムを他機種へ移植することは非常に困難です。あらかじめご了承ください。そのかわり、使いやすいプログラムが98上で実行できます。
d）プロ向け仕様のライブラリであること。本製品はあらゆる仕様を満たすよう開発していますので、内容は複雑です。シロートユーザーでは本製品は使いこなせませんので、御購入を御遠慮ください。
e）ソース公開・ライセンスフリー制度のライブラリであること。本製品を使用して販売目的の応用ソフト（EXE）を開発した場合のライブラリ組み込み使用料の請求を放棄します。安心してソフト開発に御利用ください。

なお、製品マニュアル及びソース付商品は直接販売に限定しますので、パソコンショップ等からの注文はできません。

『グラフィック・マウス・テキストライブラリ』 9,800円
『ワールド座標系グラフィックサポートライブラリ』 9,800円
『FORTRAN・MS-DOS環境変数活用ライブラリ』 9,800円

➡
①ライセンスフリー制度採用。ソースプログラム付き。
②各種C言語、FORTRAN言語別にお求めください。
③ライフボート社が発売しておりますAdvanced Run/FORTRANのグラフィック機能は、Pro-FORTRAN版の「グラフィック・マウス・テキストライブラリ」のグラフィック機能を組み込んだものです。

☆「テキスト画面高速描画処理ライブラリ」と「キーボード・データ入力処理ライブラリ」でサポートしているFORTRAN言語はMS-FORTRAN4.0だけです。他のFORTRANは、利用できません。

☆MS-DOSは、マイクロソフト社の登録商標です
☆各種言語名は、その言語を開発した会社の登録商標です

価格には消費税は含まれておりません。

開発・発売元 株式会社 日本技術ソフト開発
ソフト開発部：〒151 東京都渋谷区千駄ヶ谷5-5-10-112
TEL 03-352-1972 FAX 03-352-8397
営業時間：月曜～金曜まで 10:00～18:00
天沼企画室：TEL 03-391-1176 FAX 03-393-5471

TIS SOFTWARE PRODUCT

エンジニアリングシステム

CAD/CAMシステム
CATIS ※

会話型有限要素法解析システム
FEMAS ※

化学工学便覧情報システム
PRODECE ※

構造物骨組設計システム
ADAM ※

対話型工事費積算システム
Pub-ESS

建築物積算トータルシステム
ARC-ESS ※

データベースシステム

商標検索サービス
BRANDY

判例情報検索サービス
JUPITER

通信システム

EWSとメインフレームの分散処理接続システム
JOINT-net

異機種コンピュータ間接続システム
RAINBOW-H

ファックスメールシステム
Fair-Way

パーソナルコンピューティングシステム
OSS/NOA

ビジネスシステム

計画・意思決定支援システム
ACTIVE-DSS ※

汎用知識ベース構築推論システム
Super-BRAINS

100万語のフロッピー辞書システム
電字林

輸入ソフト

インテリジェントCADシステム
The Concept Modeller™

ソフトウェア開発支援システム

運用管理支援システム

タンデムライブラリー管理システム
CONTROL

タンデム運用支援パッケージ
TOSMAT

※本システムは、情報処理振興事業協会(IPA)の出資により委託開発されたものです。

新しい出会いを提供するTISのシステムパッケージ

“情報力”が企業戦略に不可欠な90年代において、ソフトウェアの重要性はますます高まっております。役立つ情報システムを構築するためには、統合化されたシステムが不可欠です。TISの技術には、長年にわたって培ってきた経験と実績が、時代のニーズを先取り、オンライン情報処理、ネットワーク、データベース、CAD／CAM、AI技術まで、あらゆる分野にフルに発揮されています。プランニングからメンテナンスまで、高付加価値を生み出すシステム運用、さらに機能性を高めるための提案など、充実したサービスを提供します。TISは真のシステムインテグレーターとして、次のステップへ、さらに飛躍してゆきます。

イズムをかたちに

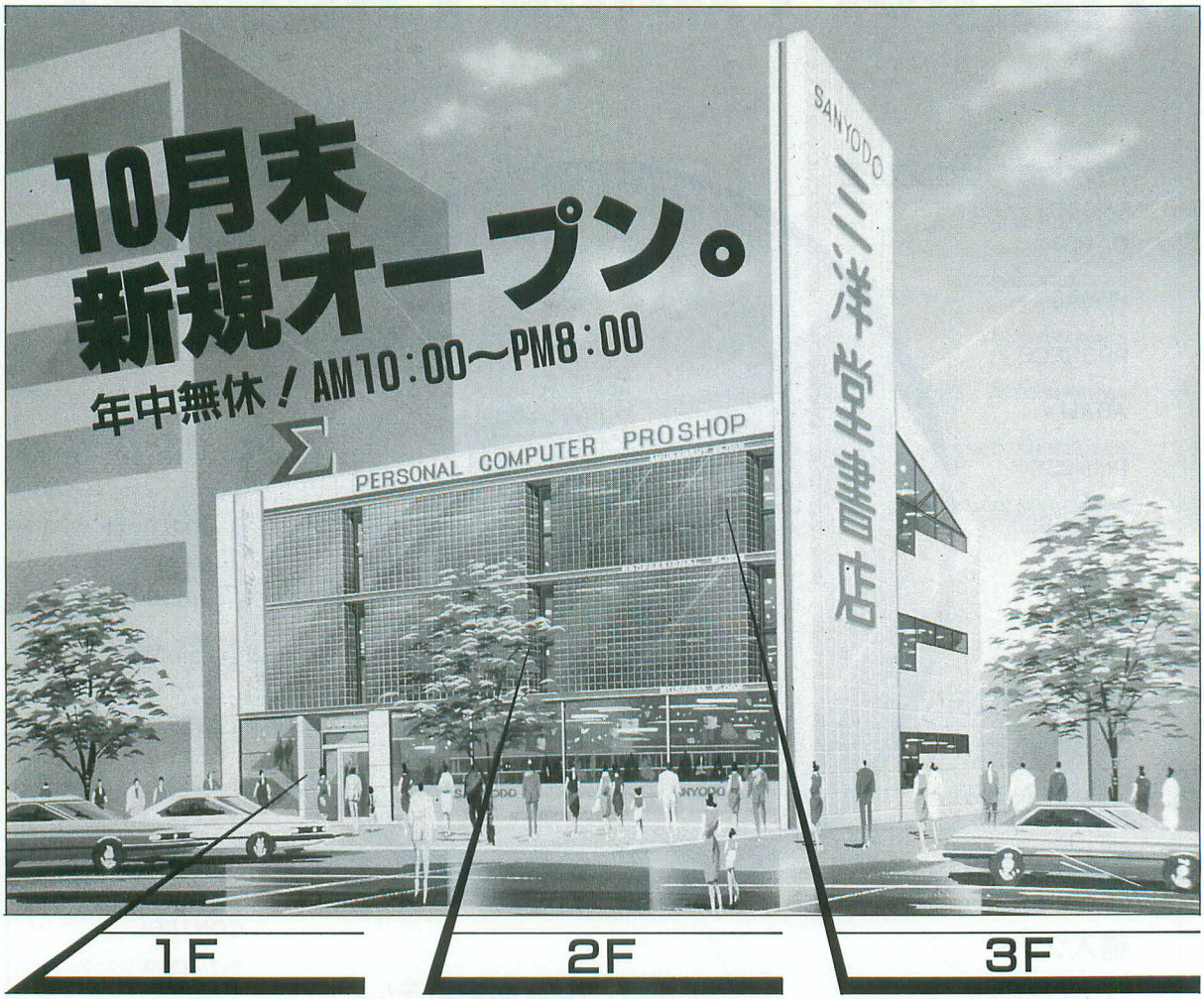
TIS

株式会社 東洋情報システム

大阪本社／〒564 吹田市江の木町１１－３０☎(06)385-0888㈹
東京本社／〒103 東京都中央区日本橋2-7-24☎(03)271-5571㈹
支　　社／東京・横浜・名古屋・京都・大阪・神戸・九州
営 業 所／全国14ヶ所・ボストン　TIS(USA)／ロサンゼルス

パソコンショップシグマ 店舗移転。

（三洋堂書店上前津店）　旧店舗より北へ100m



ビジネスを切り口にしたコンピュータ関連書籍と各種ビジネスソフト、コンピュータ関係の洋書

理工学専門書フロア

趣味の道具としてのコンピュータ＋ゲーム、各種ゲームソフト、ファミコン、ボードゲーム、Mac関連書籍とソフト、コンピュータステーショナリー

売場は旧店舗の6倍以上。より充実した品揃えで皆様をお迎えいたします。

ソフト各種：三洋堂書店各支店にてもお取扱いいたしております。

取扱い商品　■コンピュータ書籍■ビジネス／ゲームソフト各種■コンピュータステーショナリー■建築・土木・機械・電気・電子・化学・物理・数字　各専門書■ボードゲームetc.

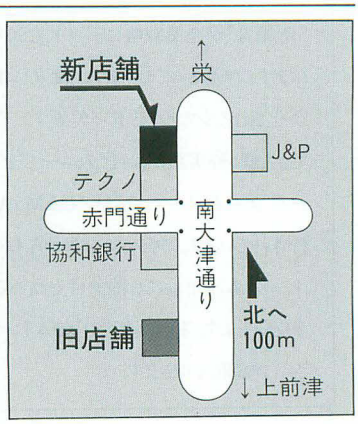
●三洋堂書店上前津店●

パソコンショップシグマ

新住所　〒460 名古屋市中区大須三丁目1006番地

☎(052)251-8334

FAX(052)251-8355



資料請求No.32

カード　　資料収集

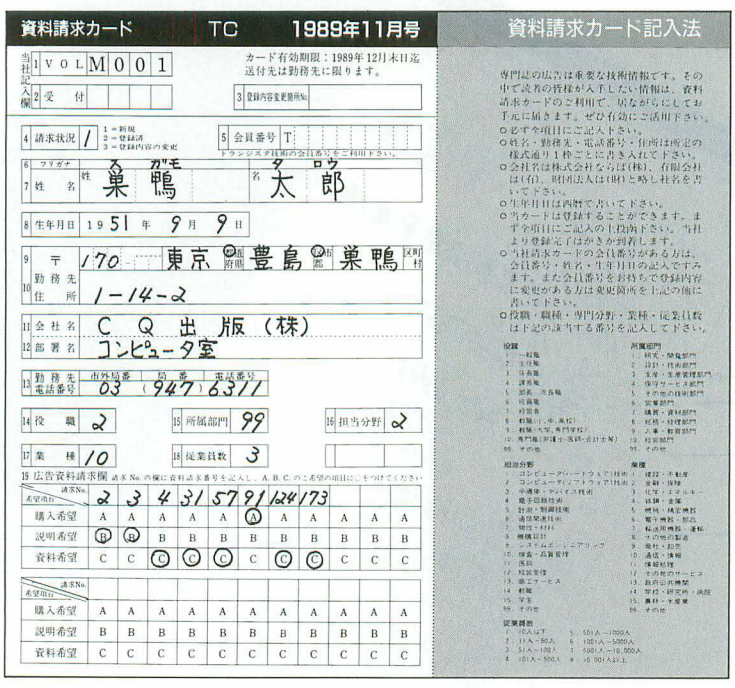
1枚＝20件

資料収集のお手伝い

トラ技コンピュータ資料請求カードご利用のご案内

技術革新は速いテンポで進歩しています。そして、その情報も同様に変化しています。そのため、資料収集が必要となります。しかし、これは時間と手間のかかる作業です。そこで、トラ技コンピュータがお手伝いします。

巻末に添付した資料請求カードをご利用下さい。トランジスタ技術が、読者に代わり、広告主にご連絡します。そして、広告主から直接読者に資料をお送りします。お忙しい読者にとって役立てたいシステムです。1枚のカードで資料収集20件、ぜひ有効にご活用下さい。


資料請求カード　TC　1989年11月号

当社記入欄
1 VOL M001　カード有効期限：1989年12月末日迄
2 受付　送付先は勤務先に限ります。
3 登録内容変更箇所No.

4 請求状況 /　1＝新規 2＝登録済 3＝登録内容の変更
5 会員番号 T　トランジスタ技術の会員番号をご利用下さい。

6 フリガナ　ナカモ　タロウ
7 姓名　姓 巣鴨　名 太郎

8 生年月日　1951年 9月 9日

9 〒 170
10 勤務先住所　東京 都道府県 豊島 区市町村 巣鴨 大字 1-14-2

11 会社名　C Q 出版（株）
12 部署名　コンピュータ室

13 勤務先電話番号　市外局番 03 局番 947 番号 6311

14 役職 2　15 所属部門 99　16 担当分野 2
17 業種 10　18 従業員数 3

19 広告資料請求欄　請求No.の欄に資料請求番号を記入し、A, B, Cのご希望の項目に○をつけてください。

請求No.	2	3	4	31	57	91	124	173			
購入希望	A	A	A	A	A	Ⓐ	A	A	A	A	A
説明希望	Ⓑ	Ⓑ	B	B	B	B	B	B	B	B	B
資料希望	C	C	Ⓒ	Ⓒ	Ⓒ	C	Ⓒ	Ⓒ	C	C	C

請求No.											
購入希望	A	A	A	A	A	A	A	A	A	A	A
説明希望	B	B	B	B	B	B	B	B	B	B	B
資料希望	C	C	C	C	C	C	C	C	C	C	C

資料請求カード記入法

専門誌の広告は重要な技術情報です。その中で読者の皆様が入手しにくい情報は、資料請求カードのご利用で、居ながらにしてお手元に届きます。ぜひ有効にご活用下さい。

○必ず全項目にご記入下さい。
○姓名・勤務先・電話番号・住所は所定の様式通り1枠ごとに書き入れて下さい。
○会社名は株式会社ならば(株)、有限会社は(有)、財団法人は(財)と略し社名を書いて下さい。
○生年月日は西暦で書いて下さい。
○当カードは登録することができます。まず全項目にご記入の上投函下さい。当社より登録完了はがきが到着します。
○当社請求カードの会員番号がある方は、会員番号・姓名・生年月日の記入ですみます。また会員番号をお持ちで登録内容に変更のある方は変更箇所を上記の他に書いて下さい。
○役職・職種・専門分野・業種・従業員数は下記の該当する番号を記入して下さい。

役職
1 一般職　2 主任職　3 係長職　4 課長職　5 部長・次長職　6 役員職　7 社長職　8 個人・自営業主　9 教職(小・中・高校)　10 教職(大学・専門学校)　11 専門職(弁護士・医師・会計士等)　99 その他

所属部門
1 研究・開発部門　2 設計・技術部門　3 生産・生産管理部門　4 保守サービス部門　5 その他の技術部門　6 営業部門　7 購買・資材部門　8 経理・財務部門　9 人事・教育部門　10 総務部門　99 その他

担当分野
1 コンピュータハードウェア技術　2 コンピュータソフトウェア技術　3 半導体・デバイス技術　4 電子回路技術　5 計測・制御技術　6 通信関連技術　7 材料・材料　8 機構設計　9 システムエンジニアリング　10 保全・品質管理　11 医療　12 経営管理　13 販売サービス　14 教職　15 学生　99 その他

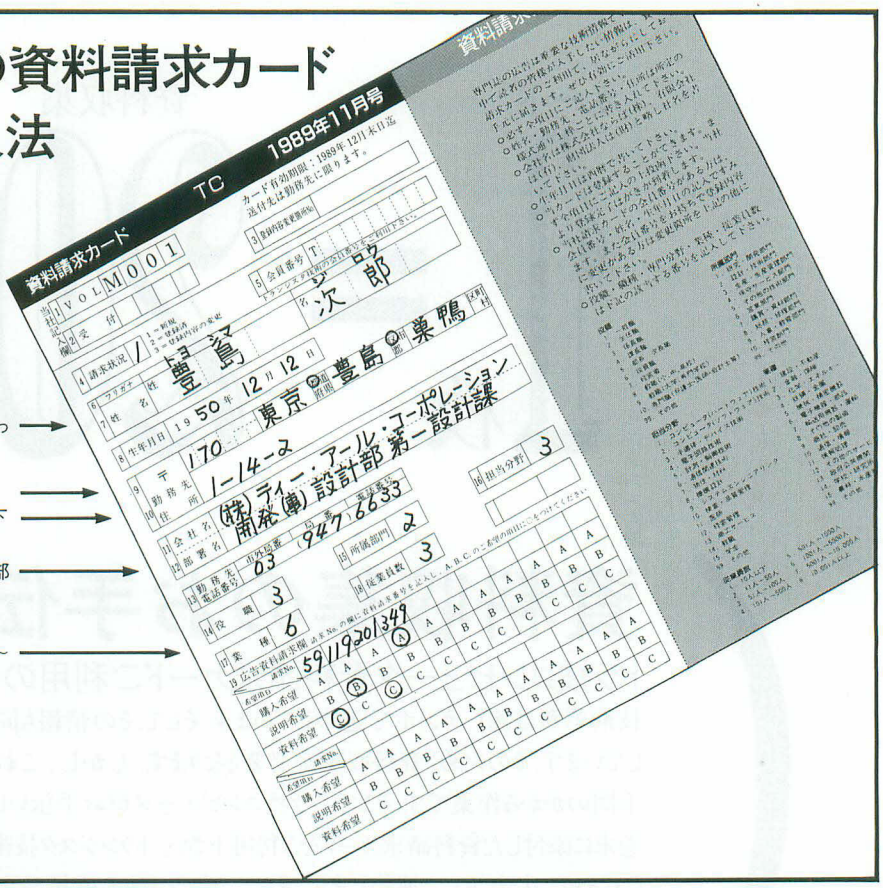
業種
1 建設・不動産　2 金融・保険　3 化学・エネルギー　4 鉄鋼・金属　5 機械・精密機器　6 電子機器・部品　7 輸送用機器・運輸　8 その他の製造　9 商社・卸売　10 通信・情報　11 情報処理　12 その他のサービス　13 政府・公共機関　14 学校・研究所・病院　15 農林・水産業　99 その他

従業員数
1 10人以下　2 11人～50人　3 51人～100人　4 101人～500人　5 501人～1000人　6 1001人～5000人　7 5001人～10,000人　8 10,001人以上

- 資料請求カードには全項目ご記入下さい。
- 学生の方は、担当分野欄に15と記入し、学校名および資料送付先住所だけをご記入下さい。
- 記入項目が不明瞭・不充分の場合、ご希望の資料が届かない場合があります。ご注意下さい。
- このカードにご記入の内容は、プライバシー保護のため、資料請求用以外は外部に公表しません。
- カードのオモテ面にある読者アンケートにもご協力下さい。
- 資料請求に関するお問い合せは下記にお願いします。

CQ出版 広告部
トランジタ技術・グループ
TEL 03・947・6311(代)
03・947・4949(直)

お勤めの方の資料請求カードご記入法



姓名を必ず分けて、フリガナをふって下さい

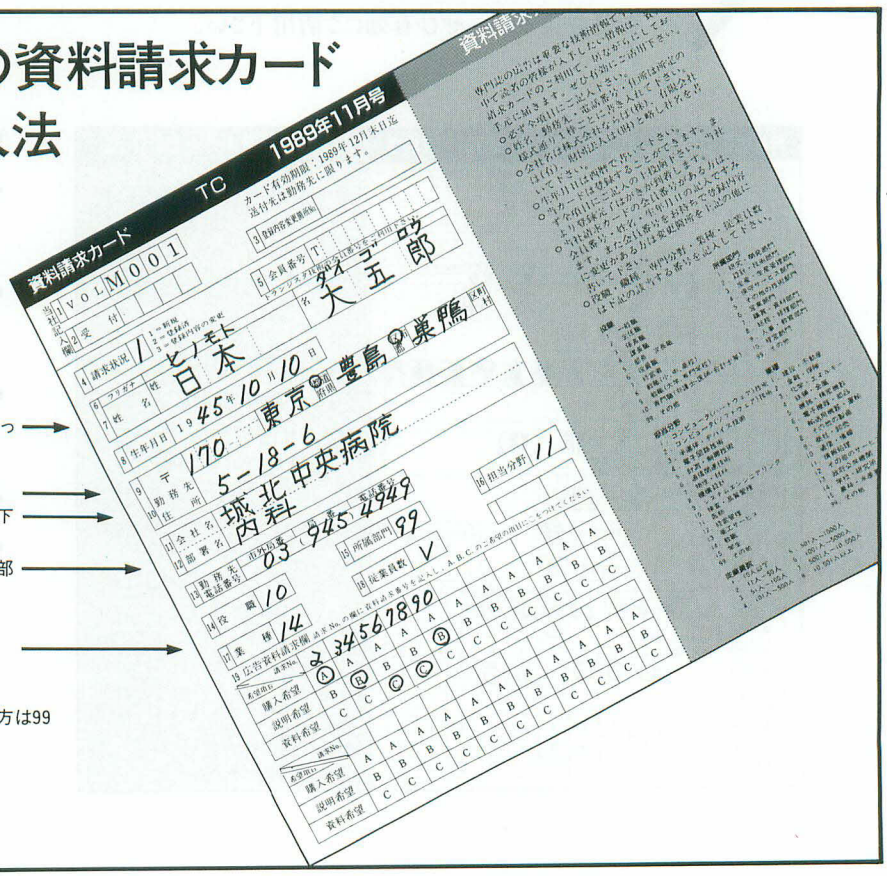
郵便番号も必ずご記入下さい
1丁目1番1号は1-1-1と書いて下さい

部署名は、事業部は(事)と略し、部課係名まで詳しく書いて下さい

従業員数は全社の従業員数で1～8の中から選んでご記入下さい

CQ出版 広告部
トランジタ技術・グループ
TEL 03・947・6311(代)
03・947・4949(直)

医師の方の資料請求カードご記入法



姓名を必ず分けて、フリガナをふって下さい

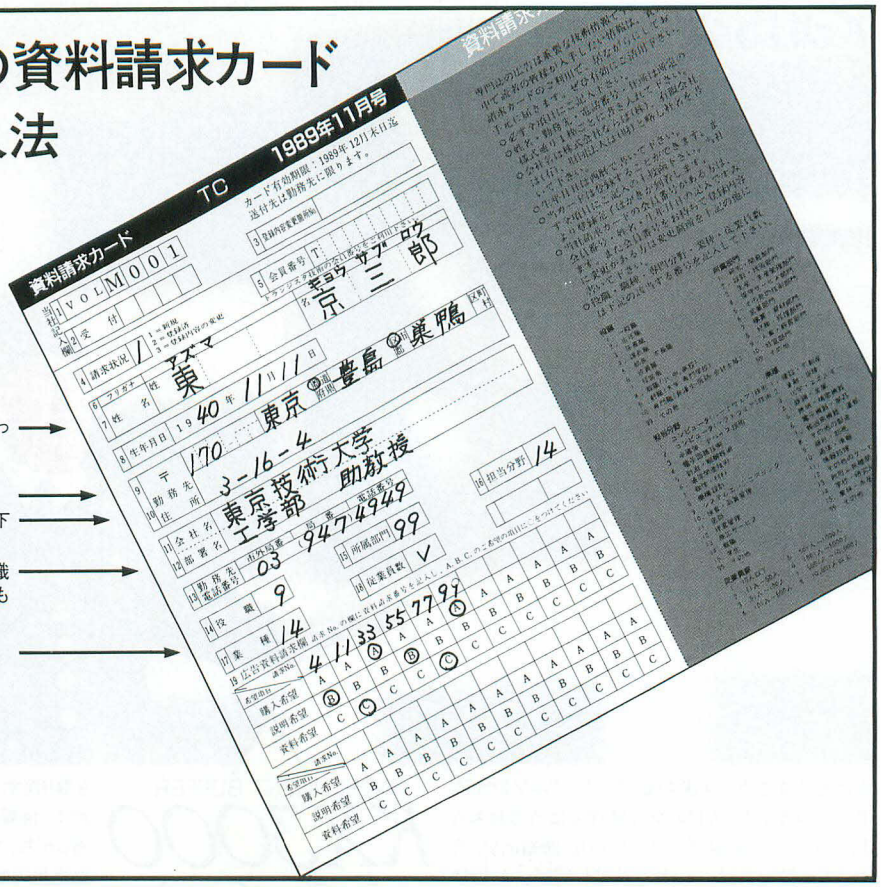
郵便番号も必ずご記入下さい
1丁目1番1号は1-1-1と書いて下さい

部署名は、事業部は(事)と略し、部課係名まで詳しく書いて下さい

14は10とご記入下さい
15は99とご記入下さい
16は11とご記入下さい
17は勤務医の方は14、開業医の方は99とご記入下さい

CQ出版 広告部
トランジタ技術・グループ
TEL 03・947・6311(代)
03・947・4949(直)

教職者の方の資料請求カード ご記入法



姓名を必ず分けて、フリガナをふって下さい

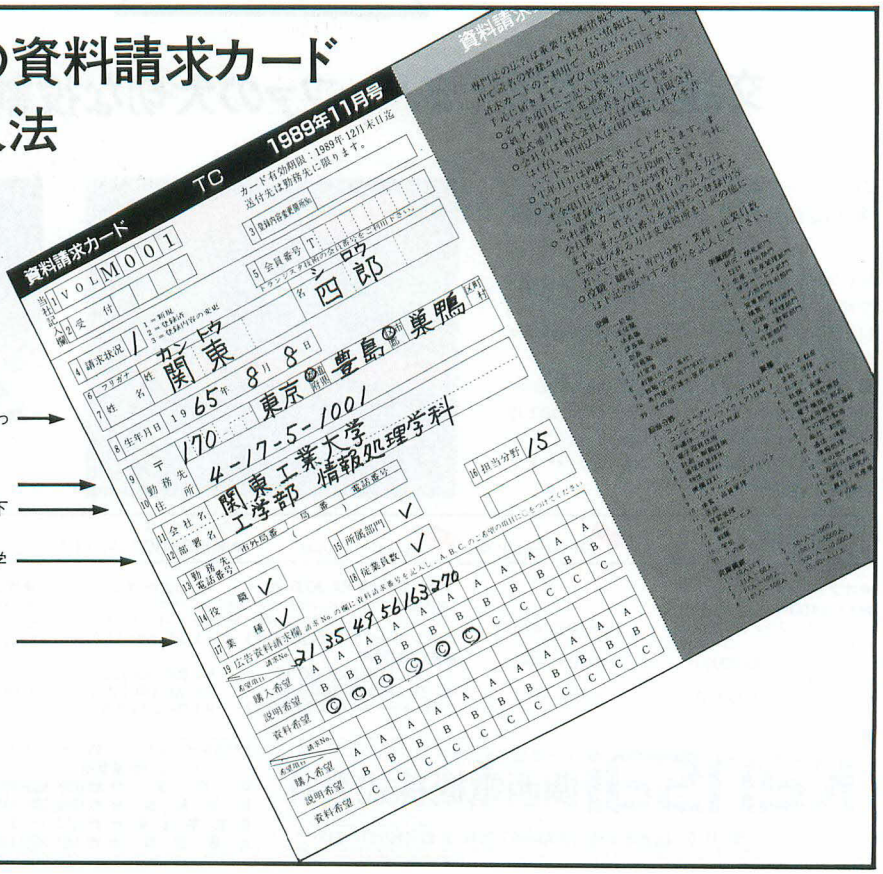
郵便番号も必ずご記入下さい
1丁目1番1号は1-1-1と書いて下さい

学校名および学部名とともに役職名(教授、助教授、講師、助手)をもご記入下さい。

14は8または9とご記入下さい
15は99とご記入下さい
16は14とご記入下さい
17は14とご記入下さい
18は無記入です

CQ出版 広告部
トランジタ技術・グループ
TEL 03・947・6311(代)
03・947・4949(直)

学生の方の資料請求カード ご記入法



姓名を必ず分けて、フリガナをふって下さい

郵便番号も必ずご記入下さい
1丁目1番1号は1-1-1と書いて下さい

学校名、学部名、学科名および学年をご記入下さい

16 15とご記入下さい
14
15
17
18 } 無記入です

CQ出版 広告部
トランジタ技術・グループ
TEL 03・947・6311(代)
03・947・4949(直)

kansai

大都会の交差点。見渡す限りクルマ、クルマの大渋滞。こうなると信号だけの交通整理では渋滞緩和もままなりません。混雑のないルートへの迂回指示や、渋滞待ち時間の把握……。臨機応変に判断し、的確な指示をだすポリスマンの努力がスムーズな車の流れ

INTELLIGENT BUFFER
KX3000 Series

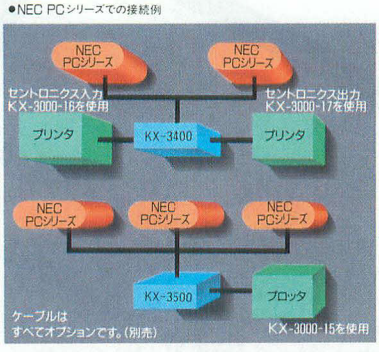
を取り戻す。今やオフィスでもコンピュータの急激な増加で、情報のはきだし口であるプリンタもデータが渋滞しがち。複数のコンピュータと複数のプリンタの有効活用のために、データの交差点にあるデータバッファの働きは、まさしく交通整理のポリスマンなのです。

交通整理もデータバッファの大切な役割の1つです。

LCDメニュー方式による
簡単設定が新しい。インテリジェントバッファは顔まで違う。

KX-3000シリーズは多入力、多出力に自動またはコマンドで対応するインテリジェントバッファです。フロントパネルのLCDメニューからコントロールする高度な機能と斬新なスタイルは、ハイエンドユーザーの要求に応えます。
●LCDメニュー方式による機能設定●入出力対応ポート/バッファ残量をLCDに表示●指定データを繰り返し印刷するコピー機能●停電時等にも動作モードを記憶しているバックアップ機能●オプションインターフェースで入出力ポートの変更可

●NEC PCシリーズでの接続例



セントロニクス仕様
●**KX-3400**：1Mバイト¥89,800/ 新発売 2Mバイト¥139,800/ 新発売 3Mバイト¥169,800

RS-232C仕様
●**KX-3500**：1Mバイト¥99,800/ 新発売 2Mバイト¥149,800/ 新発売 3Mバイト¥179,800

●オプションケーブル

■KX-3400用
PC-9801シリーズ入力用 **KX-3000-16**……¥4,000(AMP-14P/凸)
対応機種/PC-9801シリーズ・PC-286シリーズ
セントロプリンタ出力用 **KX-3000-17**……¥4,000(AMP-36P/凸)
対応機種/標準的セントロプリンタ
セントロニクス入力用 **KX-3000-18**……¥4,000(AMP-36/凸)
対応機種/16β・IF800その他
セントロニクス入力用 **KX-3000-20**……¥5,000(AMP-36/凹)
〈対応機種〉PC-98XA・N5200セントロ/FMRシリーズ・9450/バナコムMシリーズ/XI・X68000(併せて純正ケーブルも必要です。)
ジョイントケーブル **KX-3000-13**……¥4,000(DIN-13P)

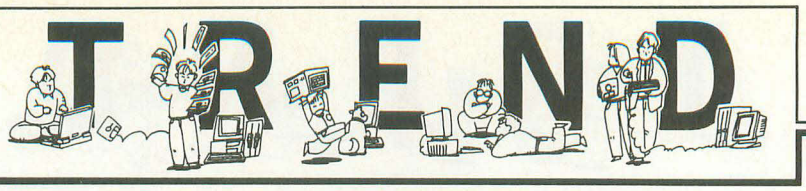
■KX-3500用
RS-232C用 **KX-3000-15**……¥4,000(DB-25凸)
RS-232C用 **KX-3000-14**……¥5,000(DB-25凹)
ジョイントケーブル **KX-3000-12**……¥3,000(DIN-8P)

●オプションインターフェイス
セントロニクス入力用 **KX-3000-01**……¥7,000
セントロニクス出力用 **KX-3000-02**……¥7,000
RS-232C入出力用 **KX-3000-03**……¥9,500

※価格・仕様は予告なく変更する場合があります。
※表示価格には消費税は含まれません。

kansai データコミュニケーションの 関西電機株式会社
本　社　〒731-01 広島市安佐南区緑井6丁目14-9 ☎(082)877-2451(代)

テクニカル・インフォメーション ☎(082)877-7912
支社・営業所
東京支社 ☎(03) 863-4021
仙台営業所 ☎(022)288-1285
名古屋営業所 ☎(052)231-9952
大阪営業所 ☎(06) 266-1168
広島営業所 ☎(082)877-3740
福岡営業所 ☎(092)474-9571
販売会社
関西電機販売㈱ ☎(03) 862-1544
関西システム販売㈱ ☎(06) 263-7991
北海道関西電機販売㈱ ☎(011)261-5590



パソコン用CADソフトは実用になるか

日本では，価格が高いこともあってCADシステムはまだ個人レベルのものではありませんが，米国ではアマチュア・レベルでもCADが使われています．ここでは，全世界に広く普及しているAutoCADとOrCADを取り上げて，CADシステムのメリットを紹介します．

パーソナル・コンピュータが普及してCAD(Computer Aided Design)が話題になっています．実際，米国ではIBM PC用のCADソフトの種類も多く非常に普及しているようですが，これからCADを導入したいと考えている人，CADとは何かを知りたい人のためにCADの魅力について話をしましょう．

● CADとは何か

CADとはたんに図面を書くシステムあるいはソフトウェアのことです．従来，機械製図や建築物図面は人間が書いていたわけです(現在もそうですが)．しかし，人間が書くことの生産性は人間により決まってしまいますし，またいくら優秀な人間でも能力に限りがあり，これを機械化したいという要求には大きなものがありました．

CADは，大型コンピュータの時代にはCADメーカの専用機を使って製品が構成されていました．パソコンもまだ8ビット・マシンの時代だったため，とてもCADをパソコンで行うことができなかったのです．

ところがIBM PCが登場したおかげで状況が大きく変わりました．パソコン上でもCADが使えるような環境が整ったのです．そうこうしている間に，AutoCADがIBM PC上のCADとして業績を伸ばしてきました．当初から機能が弱いといわれながら，オープン・アーキテクチャにすることでサード・パーティによるパワー・アップ・ツールに助けられ，多くの支持を得るようになったのです．

● AutoCADについて

AutoCADは，米国Autodesk社の製品です．メーカ発表によると全世界で20万セット，日本でも1万6千セットを販売したとのことです．もともとIBM PC専用でしたが，現在ではSUNやアポロのワーク・ステーション，日本ではPC9801，FMR，J3100などに移植されています．

現在では，英語版はRelease 10と呼ばれるものが，また日本語版はGX-IIIと呼ばれるものが最新バージョンとなっています．価格は英語版を米国で購入すれば3,000ドルですが，日本語版は日本法人であるオートデスク㈱が販売しています．定価は99万円です．米国では古いバージョンなら500ドル程度で入手できるので，この価格が高いか安いかは判断が難しいと思いますが，日本では個人ユーザは少ないため仕方がないかもしれません．

また，後でも述べますが，コンピュータとソフトだけでなく，製図を書くためのプロッタやディジタイザが必要ですので，かなりの金額が必要です．

ただし，AutoCADライクな日本製のCADソフトがAutoCADの3分の1程度の価格で販売されているので，AutoCADももう少し安くなるかもしれません．

AutoCADが普及した最大の理由は，オープン・アーキテクチャにあります．すなわち，データ構造が公開されているため，異機種間でのデータ交換が容易であり，ユーザ自身がタブレット・メニュ，ハッチング・パターン，線種などを設定することができます．そのためにAutoLISPという記述用言語も用意されています．

したがって，多くのサード・パーティが建築設計用ソフト，機械設計用ソフト，電気設計支援ソフト，データ・ベース・ソフト，データ・コンバータなどにAutoCADをカスタマイズして販売しています．そのためにAutodesk社がアプリケーションに対する責任を回避できることになります．これがAutoCADの強みでもあります．

最新バージョンのAutoCADは2次元図面だけでなく3次元図形も書くことができます．したがって，立体図形の360°回転も可能です．また，3次元図形に陰影をつけるためのAutoShadeというソフトウェアも提供されています．AutoShadeの日本での価格は15万円です．

● OrCADについて

エレクトロニクス技術者にとっては，AutoCADよりもOrCADのほうがなじみが深いでしょう．OrCAD自身電子回路設計用CADとしては世界でナンバ1の実績をもっています．もともとは電子回路を作図するだけの機能しかなく，SPICEなどの回路シミュレーション・プログラムとは異なるものです．ただし，現在では，作図だけでなく自動プリント基板作成，PLD設計，論理回路シミュレーションを行うソフトなどが用

意されています．

OrCADは，米国OrCAD社の製品ですが，日本では㈱ソーテック〔旧㈱工人舎〕が総代理店になっています．

基本となる電子回路作図用ソフトは，OrCAD/SDT IIIが最新バージョンで，価格は16万8千円です．

また，プリント基板作成用のソフトとしてOrCAD/PCBがあります．これは価格が59万8千円です．

論理シミュレータとしてOrCAD/VST(29万8千円)があります．

さらにPLDを設計するためのソフトOrCAD/PLD(14万8千円)，PLDのシミュレーションを行うOrCAD/MOD(14万8千円)などがありますが，これらをすべてそろえるとかなりの金額になります．

米国の価格と比べると日本での価格は個人が購入できるレベルではありません．

現在のところ，OrCADはIBM PCの他はPC9801に移植されているだけです．

● パソコン用CADは実用になるか

パソコン用CADは十分に実用になると思います．また，技術者ならば手書きよりもCADを使うことを当り前のように感じて欲しいと思います．ただし，AutoCADにしてもOrCADにしても，米国の価格と日本での価格にあまりにも差がありすぎるため，簡単に普及することは難しいのではないかとも感じます．

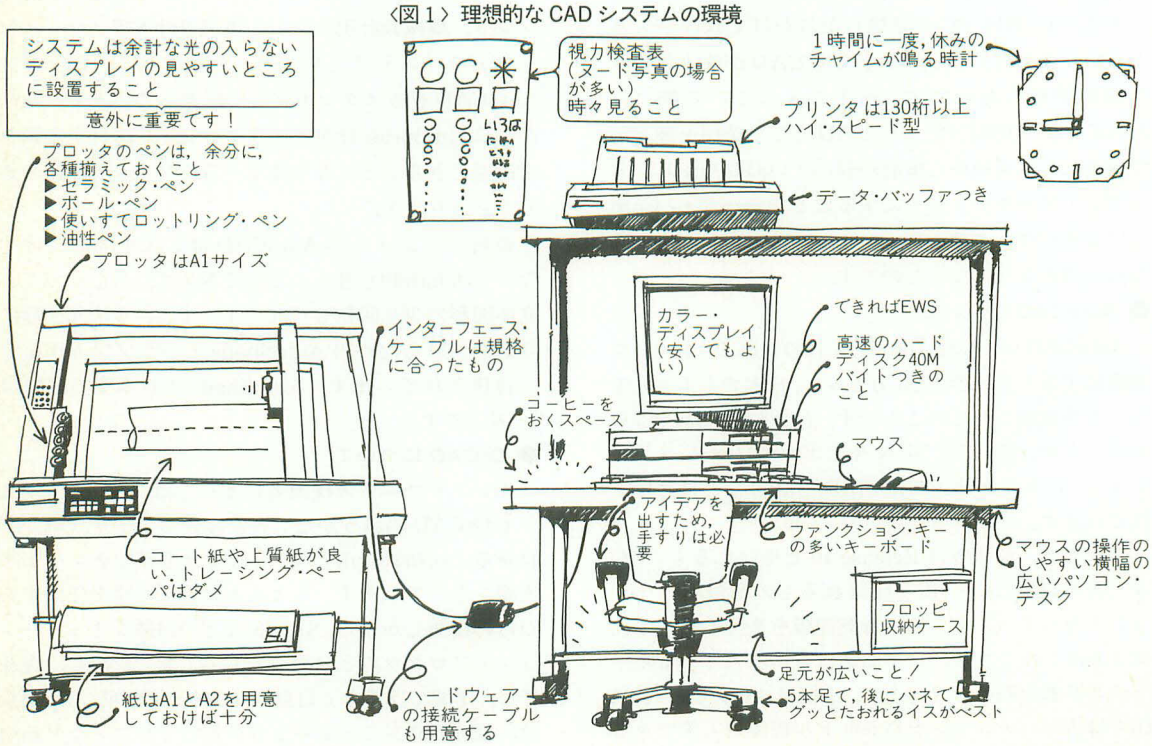
価格のことはさておき，OrCADを用いて基板設計を行う手順について説明しましょう．

現場では，性能アップとコスト・ダウン，新製品の開発などに追われ，牛どんのコマーシャルではありませんが，“うまい，安い，早い”の連続要求の毎日です．実際にCADを使うと，設計者の能力の問題である“性能アップ”を除いて，かなりこれらがうまく行きます．特に，図面の修正や性能アップに伴って，前の図面を残したり，変更箇所を記述した記録などはディスク1枚に収まるので，アップ・デート作業が格段にスムーズになります．

さらに，図面の仕上がりは，手書きの比ではありません．トレース技術の認定試験はむだなのです．書きやすさでさえ，もう手書き製図では，設計者の頭の中を描き出すクリエイティブな設計補助作業でさえも，安いパソコンCADに勝てないでしょう．

1人前の製図屋を育て，管理し，図面ケース，ドラフタなどの場所，製図用具を用意し，図面化の細かい打ち合わせをして(そのうえ，必ずといってよいほどトレース・ミスがある)でき上がる図面と，いつも使っているパソコンで，設計者自身が直接画く図とのコスト

〈図1〉理想的なCADシステムの環境



比，そして必ず変更が伴う新製品の初期図面の修正作業のコストを考えると，CADの購入費があまりに安いことに気がつくはずです．パソコンCADは，5年リースであれば，月々5万円でソフト，ハード一式が購入できます．

また，CADでのプログラミングの知識は，マクロ命令やDOSの操作を含めても大したことはありません．もちろん，ソフトの作成経験と知識があればさらによいのですが，プログラミングより図面の作成経験のほうがずっと大切です．つまり，CADを購入し，使用できるのは設計者だけなのです．単なるドラフティング・パーソンは不要です．こう考えるとCADソフトを使わない手はありません．

例として，電子回路をつくるときに必要な思想が，どこまでパソコンCADで実現できるかを，OrCADの最新バージョンで書いた簡単なシステム図面で各階層ごとに見ていきましょう．使用したパソコンは，IBM PC ATですが，機能も性能も一段上のPC9801でも同じです．

図2は，以前書いておいた回路図ファイルに入っていた，ある画像伝送装置の初段に付ける21 MHzのクロック・ライン用*LC*フィルタ回路です．これを出すのに所要時間が5分です．

本当は，4層基板のコンピュータ・ボードの実例などを紹介したいところですが，誌面の都合上，簡単な回路になってしまいました．

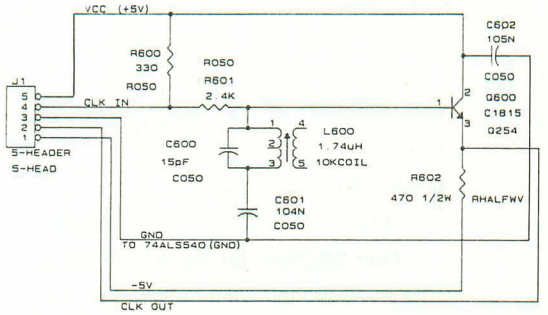
ただし，このように回路図ファイルのコレクションがたくさん集まり，ハイアラーキ(階層構造)にして書いた図面(なんてことはありません，MS-DOSのディレクトリ構造を使っただけです)は，“ブロック・ダイヤグラムを書けば，回路設計をしてくれるCAD”といってもウソにならないほどの威力があります．

次に，OrCADのERC(Electric Rule Check)機能で，設計を確認します．所要時間は2分です．

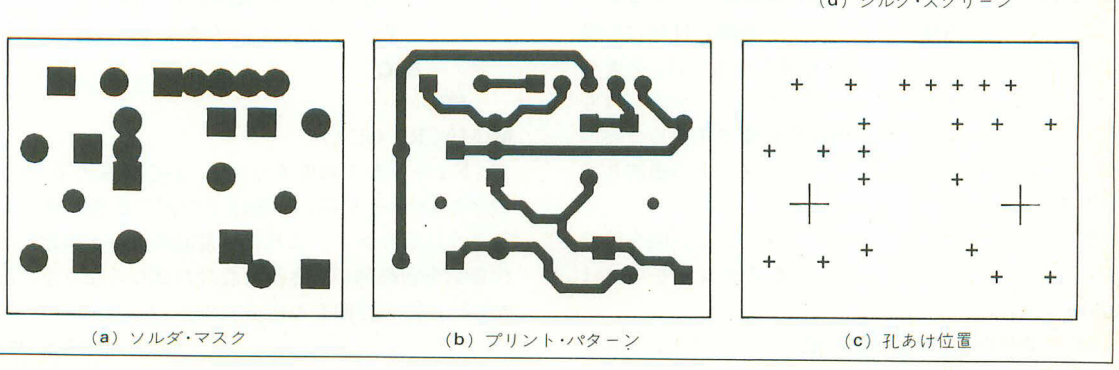
OrCADは，PSPICEと連結できるのですが，筆者はまだ試していません．ここでは，MICROCAP IIのデモ版ソフトにより動作確認をしました．最近はシミュレーション・ソフトも入手しやすくなっていますが，いずれにせよ，アナログ，ディジタル混在の大きな回路全体をシミュレートするのは不可能ですし，結果を判断しているより，基板を自動作成して作ってしまったほうが現実的だと思います(また，シミュレーション結果を，全部信じるほどのお人よしでないほうが健全です)．このための所要時間は10分です．

部品構成表は会社の部品リストのファイルから作成しました．これは，OrCADの付属ユーティリティ・ソ

〈図2〉OrCADで書いた回路例



〈図3〉プリント基板作成用パターン



(a) ソルダ・マスク　(b) プリント・パターン　(c) 孔あけ位置　(d) シルク・スクリーン

フトの説明にあるように，基本的にはワープロでMS-DOSのフロッピに書いて，OrCADの付属ユーティリティ・ソフトで自動作成できます．ただし，各自が勝手に作ってしまった部品リスト(部品名，メーカ名と型番，説明の三つが各部品に書かれている)は，OrCAD用部品リスト統合ソフトでまとめ上げたものを使用します．ここでの所要時間は5分です．

図3のプリント基板はOrCAD/PCB IIで自動配線して作ったものです．ここでの作業時間はトータルで1時間かかりました．

見積もり書はOrCAD用部品リスト統合ソフトに付いている見積もり書作成ソフトでまとめ上げたものを使用しました．所要時間は5分です．

部品発注書はOrCAD用部品リスト統合ソフトに付いている，部品発注書作成ソフトでまとめ上げたものを使用しました．この作業時間は10分間ほどです．

最後に各回路ブロックについての説明を，以前ワープロで書いたもののファイルをMS-DOSのコピー機能を使って集め，このフロッピ・ディスクに納めたものを説明書とします．

以上，すべてのトータル時間は1時間50分要しました．

各部品のメーカ名，注文型名価格，プリント基板への部品挿入製造時間や注文先の担当者名，電話番号，住所データを土台(ベース)にして，CAD回路図から生み出されるデータは，次々とほかのソフトによって結びつき，どんどんと便利になって行きます．

しかし，そのデータを作るには，コンピュータをいかに使うかがわかっているのと同時に，それぞれの部品パーツのエキスパートの助けをかりてデータ・ベースの作成をしないと，とても扱えるものではありません．このくらいになると，コンピュータとの付き合いより，部品データの収拾方法のほうが仕事になり，人間関係のほうが大事になります．

● CADシステムのハードウェア構成はどうするか

ハードウェア構成は，CADソフトの購入仕様に影響します．まず，パソコンCADを導入するには，必要なハードウェアを揃える必要があります．パソコンはもちろんですが，そのほかの機器も必要ですから一応の基本知識を身に付けておいてから，ソフトの選択と予算を組む必要があります．

最近の安価なワーク・ステーションがよい例なのですが，たとえ分解能が高くてもモノクロ・モニタはCADには向いていません．

640×350ドットの分解能が低いといっても，一番分解能が低いIBM PCでも問題はありません．図面の細かなところを見るには，ズーム・インすれば5万円程度のディスプレイでも十分にみることができます．また，必要な部分だけを拡大し，その部分をいつでも呼び出せる機能がソフトウェアについているので，作図もあまり苦労しません．ただし，プリント基板の配線や図面の全体像を見るには，さすがに高解像ディスプレイにはかないません．

ファンクション・キーは，CADの操作命令が多い機械CADの場合は15個くらいは必要です．

それぞれのファンクション・キーには，

(a) 交点から，線を書き出す．
(b) 端点から，線を書き出す．
(c) 中点から，線を書き出す．
(d) 垂線を降ろす．
(e) カーソルの近くの線分の上から，線を書き出す．
(f) 水平，垂直線のみ引く．
(g) パンニング．
(h) ズーム・イン．
(i) ズーム・アウト．
(j) 全体像を見渡す．
(k) 線を消す．
(l) 円を画く．
(m) 線をコピーする．
(n) 線の色を変える．

など，図面を書くときにはキーひとつで機能が設定されるようになっていないと不便です．

電気CADの場合，主にファンクション・キーはマクロ命令のセット・キーに使われます．概略の機能を説明すると，

M(MACRO INの意味)
　F1(F1キーで動きだすマクロ機能の設定にした)
　Find 2SC1815-O(2SC1815-Oを見つけ出せ)
　　Edit Part-Value BS(Back-Spaceの意味で，2SC1815-Oの最後のOを消すことを示す)
　　Y(2SC1815-Yと書き換え)
　　ESC
　ESC
M(MACRO OUT)

で，F1キーを1回押すだけで，2SC1815のh_{fe}ランク表示がOからYに，回路図上の記号が書き換えられるようになります．これは，部品の入手が困難になり，代替品を回路図に書き換えなければならなくなった時などには，大変便利な機能です．

〈清水　隆雄〉

光ディスクと対応ソフトの動向

音楽用として開発されたCDは，パソコンでの利用はまだまだ初期段階です．リード・オンリのCD-ROMにつづき追記型のライト・ワンスや，書き換え可能なMO(Magnet Optical)ディスクも登場しています．大容量で低速のメディアなので，利用者が使いこなす方法はこれから開拓していかなくてはなりません．

光ディスクの応用は，データ・ベースや画像記録を主として発展してきましたが，最近は技術より流行が優先するトレンド時代，マルチ・メディア，ハイパ・メディア，光ディスクはApple vs IBM戦争の最前線と化し，NeXTも光磁気ディスクをうたい文句にしています．

最初に，主な光ディスクの種別を**表1**にあげます．いずれも波長0.78 μm前後の半導体レーザ光を使い，反射の変化を記録信号として読みとります．

〈表1〉光ディスクの記録原理

種　別	書き込み原理	用　途	製品例
① エンボス	プレスによる凹凸	大量生産向き．読み出し専用	コンパクト・ディスク，レーザ・ディスク
② 穴明け	レーザで焼く	消去不可，追記専用	(いわゆる)光ファイル
③ 有機色素系	色素の化学変化，反射率または散乱変化	追記および書き換え．実用化段階	
④ アモルファス系	結晶相変化による屈折率変化	開発中	
⑤ 光磁気	熱磁気記録とカー効果による読み出し	書き換え型．実用化段階	ソニー，キヤノン，その他

● コンパクト・ディスク

音楽用のコンパクト・ディスク「CD」は，反射面の凹凸(pit/land)で信号を記録しています．CIRC*1という信号化手法による高信頼性が特色で，レコード盤同様に高速かつ大量生産が可能です．しかし，消去/書き換えはできません．

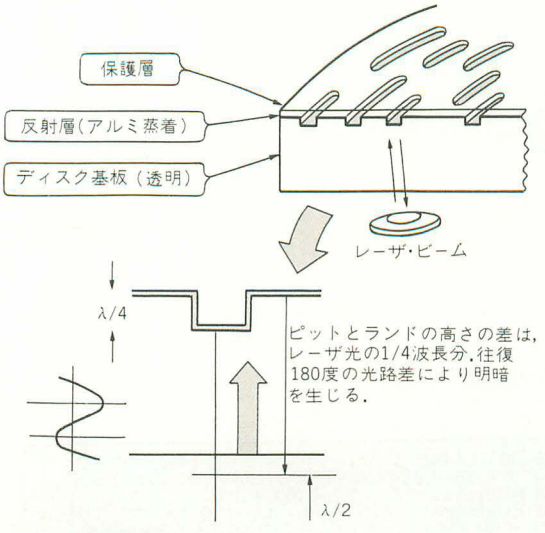
そのコンピュータへの利用がCD-ROMです．ディスク自体はCDと同一ですが，セクタ構造を追加しています．容量は560 Mバイト，音声画像などのマルチ・メディア記録に期待されています(**図1**，**表2**)．

● さまざまな書き込み方式

光磁気ディスクは，磁性体を弱磁界中におき，レーザ光で局所的に活性化して磁性を反転します．読み出しは，磁性によるレーザ光の位相回転(カー効果)を検出しています．

有機色素やアモルファス方式は，たとえば蟹をゆでると赤くなるといった原理です．不可逆性素材なら追記型(WO)ディスクとなり，双方向に変化する材質を使えば消去/書き

〈図1〉CDの読み取り原理



〈表2〉ソニー/フィリップスのコンパクト・ディスクの関連規約

規約名	内　容
レッド・ブック	コンパクト・ディスク基本仕様．読み出し専用，ディジタル・オーディオ記録
イエロ・ブック	コンピュータへのCD応用規約．データ記録方法，セクタ構成，エラー回復
グリーン・ブック	CD-I，マルチ・メディアへのCD応用規約
ブルー・ブック (CD-ROM XA)	CD-WO，追記型CD拡張規約 Microsoftとの共同提案，CD-IのMS-DOS対応サブセット

- CDという媒体そのものはレッド・ブックで規約．イエロ，グリーンはその利用方法
- その他，画像を含めたCD-Vなど応用提案は範囲外．省略．
- PHILIPSは書き換え可能なCD-RAMを提案．しかし，未完成と思える．
- SONYは，CD-ROMドライブで読み出し可能なMODディスクなども発表している．

換えも可能になります.

● 次世代光ディスク

動向としては，メーカでは3.5インチ書き換え型ディスクとか，同一ディスク中に読みだし専用エリアと書き換え可能エリアを組み合わせた Partial-ROM ディスクが注目されています.

実現すれば，3.5インチ FD とほぼ同寸で胸ポケットに入り容量100 M バイト程度，プレス量産可能かつユーザが編集可能な媒体となり，パソコンのスタイルを一変させ，光ディスクは一挙に普及するという人もいます.

一方，短波長レーザによる容量2 G バイトのスーパ CD，その高画質テレビ HDTV への応用も噂され，期待されています.

● 応用は保存と出版の2分野

書き換え型光ディスクは，大容量，メディア交換といった特色がありますが，使い方はおおすじハード・ディスクと同様です.

一方，読み出し専用ディスクの用途は大きく異なり，情報流通の手段，つまり電子出版の媒体と考えられます. たとえば CD-ROM 広辞苑のような書籍の電子化，さらに音響映像を含むマルチ・メディア出版が期待されています(図2).

● 電子出版のファイル構造

電子出版は，不特定多数の読者(ユーザ)が対象です. 読者のパソコンや OS もさまざまで，ファイル互換性が問題となります.

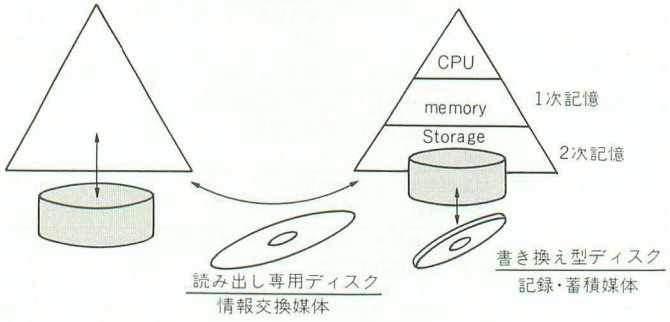
このため，OS やデバイスに対し独立な共通ファイル構造，ハイシエラ(High Sierra)フォーマットが提案されました(1986年5月*2). これは一部拡張のうえ ISO-9660 として国際標準化され(1988年)，日本でも JIS 化の途上にあります.

これにより，パソコンも UNIX マシンも1枚の CD-ROM を共用できます. さらに，CD-ROM の弱点であるシークの遅さをカバーし，巨大容量を扱えるように最適化されています.

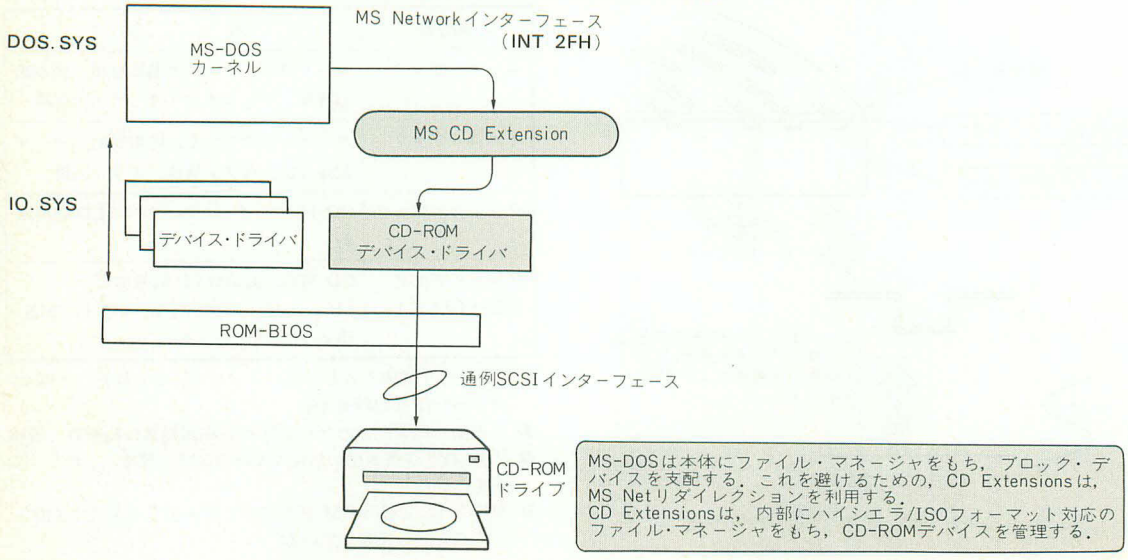
● CD-ROM/OS インターフェース

マイクロソフト社は，MS-DOS からハイシエラ/ISO フォーマットを読むためのソフトウェア，MS CD Extensions を提供しています. これは CD-ROM ドライブを(普通は SCSI ドライバ経由で)，MS-Network インターフェース上に組み込みます.

〈図2〉光ディスクの役割



〈図3〉MS CD Extensions の機構



CD-ROM 開発の入門は，このあたりからです．まずハイシエラ提案と CD Extension 仕様書を読みこなし，デバイス・ドライバを書いてみると大筋理解できるはずです．これが意外と興味深く，筆者自身もこれに没頭して，気がついたら人生が変わっていました(図 3)．

● 応用動向

最近よく登場する話題として，以下の 3 種があります．

① CD-I：コンパクト・ディスク・インタラクティブ
② CD-ROM XA；Extended Architecture
③ DV-I：ディジタル・ビデオ・インタラクティブ

① は，CD-ROM のセクタに内容を記述するサブヘッダを付け，音声，画像などの同期再生を実現する手法です(1986 年，ソニー/フィリップスが提案)．製品としては AV 家電機器に分類され，MC68000 CPU と拡張版 OS-9 が組み込まれた家庭用 CD ビデオ・プレーヤといった性格です．

② は，CD-I 同等の機能を MS-DOS などのパソコンで実現するもので，セクタ構造もほぼ CD-I と共通です．これはマイクロソフト社が加わり，IBM も採用の方向と伝えられます．

③ は，一種の画像伝送プロトコルと専用ハードウェアによる動画像ディジタル処理技術の名前です．光ディスクとは別分野ですが，製品(サンプル)は IBM PC 上の拡張ボードと CD-ROM の組み合わせで実現されています(1987 年発表，写真 1)．

● DV-I の技術

開発を始めたのは RCA 社の D.Sarnoff 研究所ですが(1983 年)，現在はインテル社が買い取り，実用化への開発が進められています．

DV-I は高速カスタム・プロセッサ，VDP 2 本が中心です(図 4)．VDP の原型は Silicon Compiler*3 で開発され，固定されたインストラクション・セットをもたず，マイクロコード自体を外部 RAM からロードして動作します．

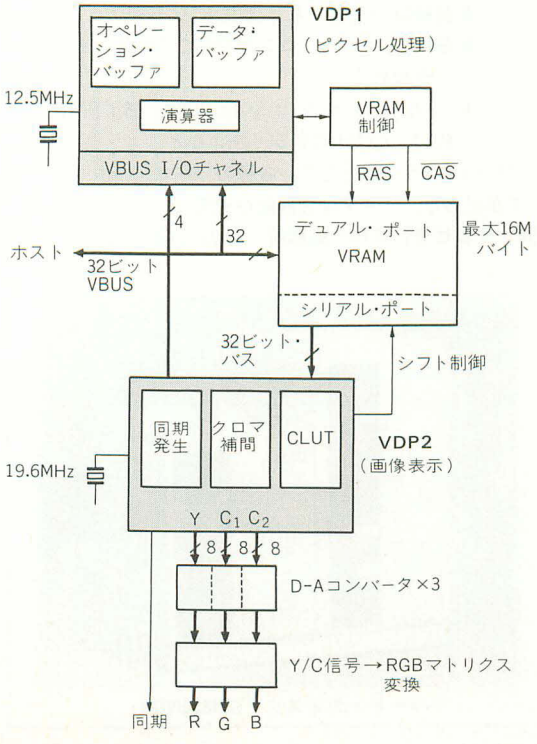
DV-I は，たとえば人物画はビット・マップ，CG 図形はコマンド描画というように表示アルゴリズムを併用し，切り替えつつ動作します．これに応じて VDP マイクロコードも入れ替え，プロセッサの動作モードをダイナミックに切り替えているとのことです*4．

● CD-ROM でビデオ記録

CD-ROM の読みだし速度は音声再生から規定され，44.1 kHz×16 ビット×2 ch≒175 K ビット/秒 程度です．NTSC や PAL 画像の再生には，この百数十倍のデータ量が必要です．

このため，DV-I は色情報圧縮およびフレーム間圧

〈図 4〉 DV-I の構成図



〈写真 1〉 DV-I の出力サンプル

〈写真 2〉 DV-I の出力サンプル
(CLUT アニメーション)

縮などにより，1フレームを5Kバイト前後まで圧縮しています．この処理は今のところインテル社自身が行います．同時に音声もADPCMで4Kバイト/秒 程度に圧縮されます．

再生および編集用ボードは，IBM PC/ATの拡張ボードとしてサンプル提供され，基本ソフトウェア・ライブラリも付属しています．動画像再生は256×240ピクセル，RGB各8ビット，1680万色，30コマ/秒です(可変)．そのデモを見ると，美しいとはいえませんが家庭用テレビに近いクオリティまで達しています．静止画像は，圧縮率1/25程度で十分に美しく見えました．

直径12 cmのCD-ROMで1時間の動画像再生ができる，というのはたしかに驚きです．しかし，DV-I本来の特色は一連のシーケンシャル再生より対話的操作などのプログラマビリティにあると思えます．たとえば，画像認識とか通信，画像データ・ベースへの応用開発も進行しているようです(**写真2**)．

〈山崎 俊一〉

◆ 参考文献 ◆

(1) 光ディスク標準化に関する調査報告，光産業技術振興協会．
(2) Digital Video Interactive Technology ほか，Intel Corporation．

(*1) CIRC Cross Interleave Reed-Solomon Coding：媒体の傷，欠損などのバースト・エラー対策として，データを並べ変えたうえで，ガロア有限体上の演算，リード・ソロモン符号を添え，エラー検出，回復する．CDはこれを2段繰り返している．

(*2) High Sierra はネバタ州の地名．ここにSONY，PHILIPS，Microsoft，Appleなど十数社が集まり，規約を作成した．

(*3) Silicon Compiler System：カスタム・プロセッサのマスクを自動設計するソフトウェア・システム．

(*4) VDPの速度は12.5 MOPS，つまり1250万操作/秒とされる．ビット幅固定のインストラクションではないから，操作とオペレーションで数える．ただし，そのオペレーション・コードは特許とされ公開されていない．

INFORMATION

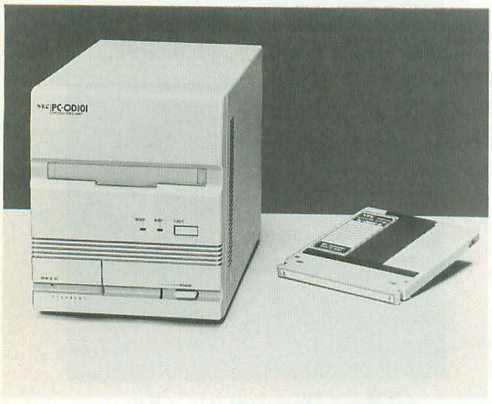
● パソコン用光ディスク・ユニット

【型名】PC-OD101　【価格】450,000円

【特徴】▶書き込み/消去可．平均シーク時間68 ms
▶片面約300 M/両面約600 Mバイト(パーティションの最大は64 Mバイト)
▶光ディスクはカートリッジ式(単価 30,000円)
▶SCSIインターフェース(PC9801-55カード)

◎ PC9801シリーズ用の大容量リムーバブル光ディスク．MS-DOS3.3+サポート・ソフトで使いがってはハード・ディスクと同じ．

【問い合わせ先】日本電気㈱　☎03-452-8000



光ディスク・ユニット「PC-OD101」

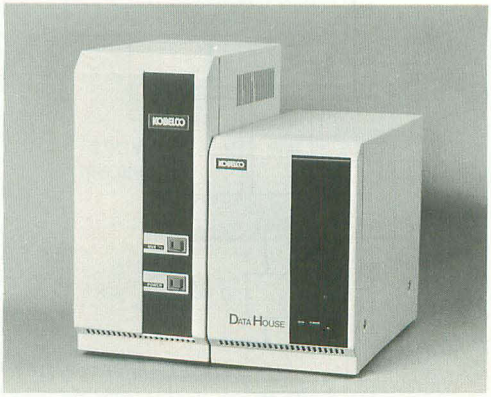
● パソコン用大容量ハード・ディスク装置

【型名】DH320ND　【価格】1,250,000円

【特徴】▶容量 320 Mバイト(最大6台接続可)
▶接続時の常駐メモリは15Kバイト
▶最大56台のパソコンを接続でき，ネットワークが組める
▶つながるパソコンはPC98，55，FMR，PANACOMの各シリーズなど

◎ ファイル・サーバとして，各種のパソコンのネットワークが組める．データ保護機能が充実している．

【問い合わせ先】㈱神戸製鋼所　☎03-5476-9222



ハード・ディスク「DH320ND8」

80386の仮想86モードを使うとできること

最近のアプリケーション・プログラムは，PC9801のメイン・メモリ640Kを使いきるものも多くあります．80386の仮想86モードを使ったソフトのひとつとして，メイン・メモリを広く使うためのデバイス・ドライバ類があります．

1985年に発表された32ビットCPU 80386は，4Gバイトの物理アドレス空間をサポートでき，下位CPUのエミュレートができるモードをもっています．現在パソコンで利用者の多いのは80286ですが，本来のプロテクト・モードと呼ばれる機能は使われず，大半が8086実アドレス・モードで利用されています．

80386では，**図1**に示すように，8086, 80286, 80386, 仮想8086モードの合計4モードで動作ができるように設計されています．起動時には実アドレス・モードなので，高速8086マシンとして，今までのMS-DOSなどのソフトをそのまま動かすことができます．80286モードを使えばOS/2も動きます．しかし，80386のプロテクト・モードを利用できるソフトウェアは，現在ほとんど市場に出ていません．

四つ目の仮想86モードと実アドレス・モードの違いは，仮想86マシンが動いているときに，記憶保護，仮想記憶および特権チェックなどのプロテクト・モードで使われる機構が生きていることです．8086では実際の物理メモリをアクセスするには，セグメント＋オフセットでアドレスが決定します．仮想86モードは，プロテクト・モードの中で動作しているのでページングの機能も使えるため，実際の物理アドレス空間がばらばらでも，論理的に連続した空間として利用できます．

また，仮想86モードでは，入出力ポート，割り込みベクタ・テーブルももった1Mバイトの空間が複数得られます．これを利用したのがWindows/386です．

ほかにも仮想86モードを使うことで様々な応用が考えられますが，ここでは最近製品化されている代表的なソフトについてみてみます．

● EMSへの対応

主記憶の640Kバイト以上のメモリ空間を利用するためにEMS 3.2が作られ，さらにマッピングに柔軟性をもったver4.0では，32Mバイトまでの拡張メモリが使えます．80386のページングの機能を使うことで，1Mバイトの最上位番地より上にあるメモリ部を，EMSのための拡張メモリとして利用できます．

● XMS(eXtended Memory Specification)

増設メモリの効率的利用を図るためのXMS規格の中には，HMA(High Memory Area)とUMB(Upper Memory Block)さらに増設メモリ・ブロックの三つの利用方法が規定されています．UMBは後述しますが，利用されるのはHMAが主体です．これは，**図2**に示すように，アドレス線A_{20}(21ビット目)信号を利用することで，1Mバイトの上64Kバイト足らずを利用する方法です(80286でも可)．

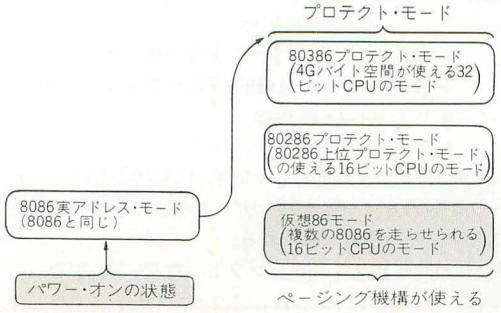
● UMBへの対応

このUpperとは**C0000H**から**E0000H**の本来ROMが配置されるべき領域ですが，普通はほとんど空いています．仮想86モードでは，ページング機能を利用して，この物理アドレス128Kバイト分をRAMがあるように使うことができます．

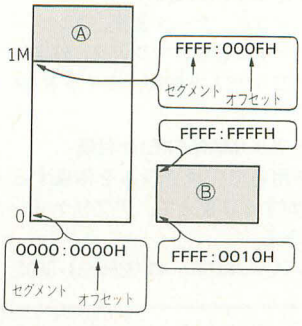
● バンク・メモリへの対応

メイン・メモリの上位128Kバイト(512K～640Kの間)を窓としてバンク・メモリとして利用する方法が日本では一般的です．プロテクト・モード用の1Mバイトから上位の部分を使い，仮想バンク・メモリ方式

〈図1〉80386の各モードとその関係



〈図2〉HMA(High Memory Area)の構成



8086のアドレス・ピンはA_0～A_{19}の20ビットなので，セグメントが**FFFFH**のとき，オフセットの値によっては1Mバイト以上のメモリをアドレッシングしたらⒶ，最上位のビットが捨てられて，ラップアラウンドしてⒷの位置にきてしまう．80286/386ではA_{20}以上のピンが出ており，リアル・モードでもこのピンを有効とすることができるパソコンが多い．

を実現できます.

このような機構が考えられた背景は，8086 実アドレス・モードのプログラムが主流なのに，1 M バイトの空間ではたりなくなってきたことが理由です.

デバイス・ドライバ，常駐型(TSR)やバッファ領域が増えると，利用できるメイン・メモリは 300 K バイトを切る状態がおこります.

そこで，今利用されていない ROM 領域に，デバイス・ドライバやバッファ(**config.sys** の中の **Buffer**＝で指定される部分)の実体をおくことで，40～100 K バイトの空間が開放されます．もちろん，使われている ROM 領域に重ならないようにメモリ空間を配置しなければなりません．これが UMB の機構です.

以上のことを PC9801 のメモリ・マップに示したものを図 3 に示します.

● 市販のソフトウェア

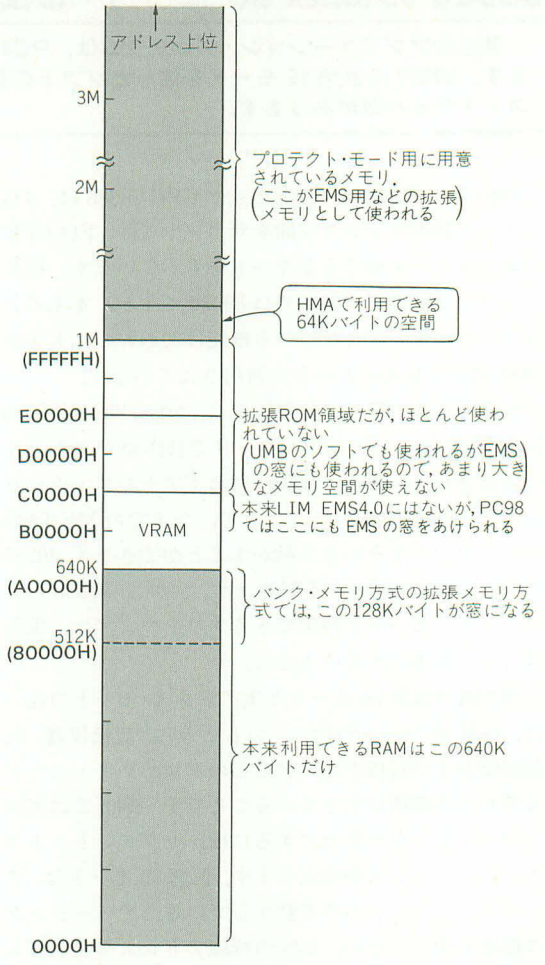
市場にこの仮想 86 モードを活用した EMS ドライバを最初に出したのがメガソフト社で，製品名は「MEMORY-PRO386」です．サポートしているのは，仮想 EMS，仮想バンク・メモリ，仮想 UMB，XMS (HMA)です.

次にメルコ社から「MELWARE Ver.4」が発表されました．サポートしているのは，仮想 86 モードの EMS(ハードウェア EMS と併用可)，バンク・メモリ・ボード，UMB(バッファーズ拡張のみサポート)，XMS です.

また，アイ・オー・データ機器社から「IOS-10/386」が出荷予定です.

〈編集部〉

〈図 3〉 PC9801 のメモリ・マップ



◆ 参考文献 ◆

(1) メガソフト，MEMORY-PRO386 ユーザーズマニュアル.

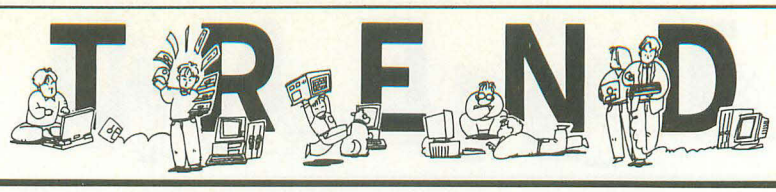
INFORMATION

● C 言語の関数ライブラリ

【商品名】パワーアップキット(C-FUNCTION98 用)
【価格】業務用：55,000 円，個人用 12,000 円
【特徴】▶グラフィック，マウス，データ変換，コンソール，文字列，ファイル管理，プリンタ，外字処理，RS-232C，日本語入力制御，ウインド・メニュなど
▶オート・マニュアル(**MAN.EXE**)が付属
◎ PC9801 上で C 言語を用いてプログラムを作成するとき，ベーシックな関数がすぐに使えて，アプリケーションの開発時間を短縮できる.
【問い合わせ先】㈲原田システム設計　☎ 0286-34-5605

● パソコン間通信ソフト

【商品名】PC ブリッジ　【価格】58,000 円
【特徴】▶ MNP クラス 5 モデムに対応―エラー・フリー，4800 ボー使用可
▶文書，データのすべてを送受信できる
▶自動ダイヤル，自動送受信ができる自動運転
▶リモート DOS 機能
▶セキュリティ機能
◎ PC9801 シリーズ同士をつなぎ，信頼性の高いネットワークが組める．バイナリ・データも送れるので，社内の LAN に近い環境が得られる.
【問い合わせ先】㈱インターソフト　☎ 03-293-3338



ハイパ・メディアの現況

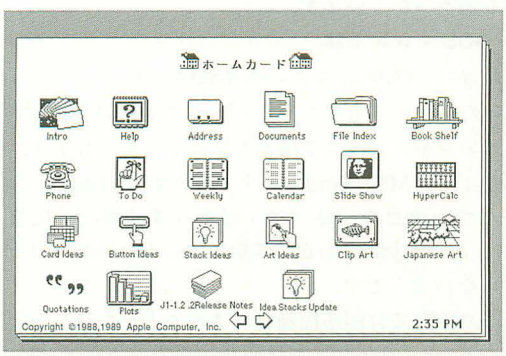
パーソナル・ユースのコンピュータのOSを分類すると，DOSマシン，Mac，Unix，その他となり，そのMac上では最近マルチ・メディアが進行しています．音・絵・文章・データなどをまとめて扱うことをめざしています．その先鋒がハイパ・カードです．

近年，パソコンの処理能力が飛躍的に増大するにつれて，テキスト，グラフィックス，サウンドなどを自在に組み合わせて，ユーザに様々な情報を提供する「ハイパ・メディア」的な用途が実用段階に入ってきました．このような方向を目指したシステムとして，すでに実際に広く普及しているものに，Apple社のパソコンMacintosh用の「HyperCard」があります．以下，このHyperCardがどのようなシステムであり，どのように利用されているのかについて，簡単に紹介します．

● HyperCardの特徴

HyperCardはカード単位で情報を管理する「情報提供/構築システム」です．HyperCardでは，Macintoshパソコンの9インチのモニタ画面(モノクロ512×342画素)全体を，ちょうど一枚の白紙のカードと見立てます．このカード上には文章，図表，イラスト，コメントなどを自由に書き入れておくことができます．こうしたカードをたくさん集めておいて，情報を管理したり，提供したりするわけです．HyperCardでは，このようなカードの集まりのことを「スタック」と名付けています．

HyperCardではまた，カード上に「ボタン」を配置することができます．ユーザがマウスを使ってカード上のボタンを押すと，たとえば関連する情報を収めたほかのカードに飛んだり，サウンドを鳴らしたり，簡単なアニメーションを表示したり，といった動作が引き起こされます．



ハイパーカード・システム標準のホーム・カード．画面に並んでいるアイコンはすべてボタンになっており，これらをマウスでクリックすると，様々なほかのハイパーカードのスタック（ヘルプ，住所録，カレンダ，メモ帳など）が起動される．

カードとカードの参照関係は自由につけておくことができ，どのカードからどのカードへでも飛ぶことができます．ユーザは，カード上のボタンをマウスで操作していくだけで，様々なカードに記載された情報を勝手気ままに読み漁ったり，新しく書き加えたりすることができるわけです．

● HyperTalkというプログラム言語をもつ

ちなみに，ボタンに対する反応などの動作は「HyperTalk」というオブジェクト指向的な特長を備えたプログラミング言語を使って記述します．といっても，HyperTalkは別に難しいものではなく，100語程度の平易な英単語を予約語としたシンプルな言語で，おそらくBasicで簡単なプログラミング経験をもった人ならすぐに馴染めると思います．また，スタックを構成するためのカード，ボタン，グラフィックスなどといった部品も単体で流通しているので，ゼロから作るのが面倒な人は，これらを組み合わせるだけで，ユーザがまったく新しいスタックを作り出すことができます．こうした手軽さもHyperCardの大きな魅力の一つです．

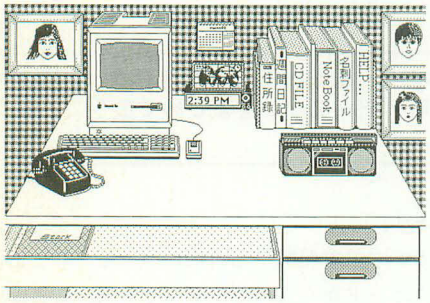
● HyperCardの歴史

HyperCardが発表されたのは1987年の夏ですが(日本語版の発表は1988年夏)，まったく新しい未来のパソコン利用の方向を具体的に指し示したシステムとして，発表と同時に非常に大きな反響を呼びました．

HyperCardは，Macintoshを購入するとシステム・ディスクとともに標準で添付されています(国内購入のMacintoshには日本語版HyperCardが標準添付されている)．したがって，HyperCardのスタックとして提供された情報は，Macユーザなら誰でもMacを使って読む(見る，楽しむ)ことができ，また反対にMacユーザなら誰でもHyperCardのスタックを作ってほかのMacユーザに提供することができます．このおかげで，Mac関係の電子ネットワークなどでは，様々なパブリック・ドメインのスタックが交換されています．

また，HyperCardを使って作成したスタックは，配布に関しても販売に関しても いっさいのライセンス

TREND



ユーザが独自に作成したホーム・カードの例（左）．

昨年度のスタックウェア・コンテストに入賞した妊娠知恵袋というスタックの一画面．現在の日時に合わせて，予定日までの残り日数や，アドバイスが表示される．また，画面上部に並んだボタンをクリックすると妊娠時に関連したさらに詳しい情報がいろいろと表示される（Nifty-serve Macusデータ・ライブラリより）（右）．

が無料で公開されており，これに基づいてHyperCard dで作成したスタックを 製品として生産・販売することも行われています．

● HyperCard の利用

それでは具体的にHyperCardはどのように利用されているのでしょうか？

まず標準添付のHyperCardシステムには，住所録やカレンダ，スケジュール手帳などといった「電子システム手帳」を実現するスタックがたくさん付属してきます．またHyperCardが登場した1987年夏のMac-World Expoでは，会場案内システムがHyperCardで作成されました．このほかに，Macintoshのセールス・マニュアル，手話の練習システム，会社案内（年次報告書），自転車修理のパンフレット，子供向けの電子絵本など，様々なスタックが出回っていますが，やはり教育用や情報照会システムなどの用途が多いようです．

また，HyperCardをユーザ・インターフェース面でのフロント・エンドとして利用する試みも行われています．たとえば給与管理システムなどを構築する場合に，データ・ベースの核の部分はほかのデータ・ベース言語やC言語などで記述しておいて，ユーザ・インターフェースの部分はHyperCardを使ってデザインの専門家に作らせます．

そして両者を合体させれば，ユーザにとって見栄えがよく 操作しやすいインターフェースを短時間で効率よく作成することができます．HyperCardを使ってあらかじめユーザ・インターフェース部分だけをプロトタイピングしておけば，顧客に具体的な設計案を提示することができますし，後で変更があった場合にも対応が容易になるでしょう．

もちろん，理想的なハイパ・メディアという面から見た場合には，HyperCardにも欠点はいろいろとあります．たとえば，① カードの表示面積が小さい，② 一度に1枚のカードしか表示できない，③ モノクロである，④ 音声と動画の面が弱い，⑤ HyperTalkの実行を行うインタプリタが遅い，などです．けれども，これらはMacintoshの標準的なハードウェア仕様を考える場合には仕方のないことでしょう．

● HyperCard の将来

現在のところ，Apple社ではコンパイラや高速ランタイム・ユニットのようなものを準備中であるようです．また，今年の春にはSilicon Beach社というサードパーティのソフト・ハウスから，HyperCardを大幅に機能強化した上位互換製品SuperCardが発表されています．このSuperCardでは，カードの大きさが無制限で，カラー表示ができるなど，上記の欠点のほとんどが解消されています．Mac II以上の機種では大きな威力を発揮するものと期待されています．

HyperCardではグラフィックスなどのデータがどうしても大きくなりがちなので，Apple社ではCD-ROMの活用を進めていく意向のようです．実際に，Apple社が配布するTechnical Noteという技術者向け資料は，来年以降はCD-ROMに収めたHyperCardのスタックとして供給されるようです．

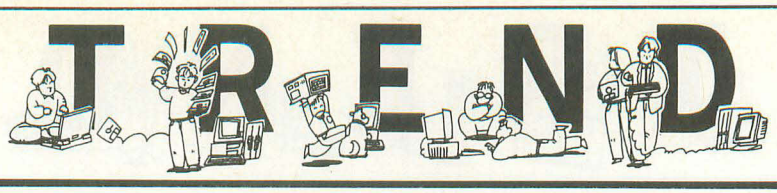
また，MIDIを利用して外部音源を制御したり，LDを併用してHyperCardからコントロールすることなども試みられています．

● DOS マシンでは

最後に，PC9801などの国産機種も32ビット時代に入った以上，まもなくハイパ・メディア的な利用がひとつのジャンルとして確立してくることでしょう．今のところ，MS-Windows上で動作する「Guide」などいくつか注目すべきソフトはありますが，まだまだMac上のHyperCardに匹敵するような広がりはもっていないようです．

この分野では国産機種はいずれも大きな遅れをとっていますが，具体的な対抗製品が出る場合には やはりHyperCardがひとつの目標になるのではないでしょうか．案外，この分野ではファミコンなどがいちばん近道にあるかもしれません．

〈廣野 幸治〉



Xウィンドウがもたらす世界

パソコンもネットワーク・システムを構成するようになるとウィンドウ・システムが必要になってきます．パソコン用としては，最初からウィンドウをもつMacintosh，MS-DOSマシン用のMS-Windowsなどがあります．ここでは，UNIXマシンの世界で標準となりつつあるXウィンドウを紹介します．

● X Window Systemとは

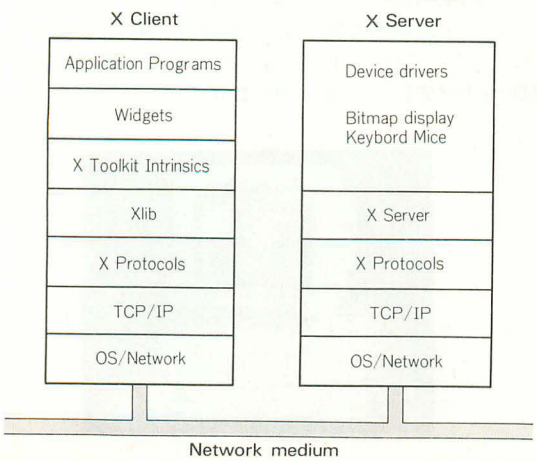
X Window System(以降，Xウィンドウ)は，M.I.T.のRobert Scheiferらが中心となって開発したパブリック・ドメイン・ソフトウェアで，現在はほとんどの主要なワーク・ステーション上で稼動しています．

Xウィンドウといっても，1987年以前に用いられたXウィンドウVersion 10とそれ以降のXウィンドウVersion 11では，ネットワーク上を流れるプロトコルが異なっています．両者のプロトコルを交換するプログラムも用意されていますが互換性は高くありません．現在流通しているXウィンドウはX11R3と呼ばれるもので，M.I.T.から配布されるテープに含まれるソースの量は60Mバイトを超える巨大なシステムです．

従来のウィンドウ・システムはOSやハードウェアに近いレベルで組み込まれているため，ハードウェアごとに異なるウィンドウ・システムが使われていて，それがアプリケーション・プログラムの移植性を低下させていたのですが，Xウィンドウではウィンドウ・システムの動作をアプリケーション・プログラム側(Client)のリクエストとそれを解釈，実行するウィンドウ・サーバ動作側(Server)に分離してモデル化(Client-Server model)することにより，機種依存性を最低限に抑える構成になっています．

Xウィンドウではウィンドウの操作やキーボード，マウスとのインタラクションを行う部分をサーバ側とよび，その機能を使うアプリケーション・プログラムをクライアント側と定義しています．

〈図1〉 Xウィンドウ・システムの構成



Xウィンドウは無償であることと主要なワーク・ステーションでサポートされていることからアメリカ国内の大学で広く受け入れられ，その後ソニーのNEWSシリーズのように日本のワーク・ステーション・メーカも採用する動きが出てきています．現在では，ビット・マップ・ディスプレイを装備した多くのEWSに移植されています．

Xウィンドウは次のような特徴を備えています．

- ▶サーバとクライアントの通信はTCP/IP(*1)を用いるため，サーバとクライアントが物理的に分離しているハードウェアでもよい．したがって，分散計算環境にうまく適合する．
- ▶プログラマ・インターフェースのレベルが各種用意されていて，目的にあったインターフェースを採用できる．一般的なインターフェースはC関数ライブラリである**Xlib**であるが，上位のインターフェースとして**X Tool Kit Intrinsics**，**Widgets**といったアプリケーション指向のプログラム・インターフェースの開発が進行中である．
- ▶他言語へのマイグレーション(移植，移行)が可能である．TCP/IPとインターフェースできる言語であればCに限らず容易にインターフェースできる．この例としてCommon Lispインターフェースである**CLX**がXソース・テープに含まれている．
- ▶アプリケーション・プログラム側にはウィンドウ操作の手続きが含まれるわけではないので実行イメージがコンパクトになる．
- ▶マルチフォント，カラー端末がサポートされる．
- ▶ウィンドウに対するグローバルな操作はウィンドウ・マネージャと呼ばれるバック・グラウンド・プロセスが行うため，アプリケーション・プログラム側はそれを意識しなくてもよい．

等々，いま考えると当たり前のように思えますが，これらの仕組みをうまくシステム化して実装した点，そしてそれを無償で公開する奉仕の精神は賞賛に値します．

Xウィンドウが公開される以前の代表的なウィンドウ・インターフェースといえばSun MicrosystemのSunView(Sun Tools), Apollo ComputerのAGESなどですが，それらはそのファミリ内で閉じたものであり，当然ながら安定性と品質は保証されていますが解放的なものではありませんでした．

初期のXウィンドウ(Version 10)は仕様的にも小さなものであり，既存のウィンドウ・システムと比べた場合，外見的に見劣りしたにもかかわらず多くのユーザを獲得できたのは，それがハードウェアやOSの枠を越えたパイプを提供したためであったと思います．

最近ではメーカが正式にサポートするウィンドウ・システムでもXウィンドウを無視できなくなり，Xウィンドウのプロトコルを受け付けたり，あるいはDECのようにXウィンドウを正式にDEC Windowsとして採用したりする動きもあります．

● **Xウィンドウのプログラム・インターフェース**

前述したように，Xウィンドウはいくつかのレベルのプログラム・インターフェースを解放しています．**図1**がそのモデルで，Xウィンドウを用いたアプリケーションを作成するといってもいくつかのインターフェースが可能であることがわかると思います．

Xウィンドウが規格として厳密に定義しているのは，実はXプロトコルと呼ばれるTCP/IPを流れるデータ形式です．したがって，最も自由度の高いアプリケーションはTCP/IPでじかにXサーバを接続するかもしれません．Xウィンドウはこのレベルで規格化されているためにOSや言語処理系を越えた接続が可能になります．

また，意外に思えるかもしれませんが，Xウィンドウではサーバとアプリケーション・プログラムが同じマシンで動いていても見かけ上はTCP/IPで接続されています．そのくらいサーバとアプリケーション側は明確に分離されているわけです．

プログラマ・インターフェースとしてXウィンドウVersion 10の頃から普通に用いられていたのが**Xlib**インターフェースで，これは個々のXウィンドウのプロトコルに対応するCの関数のライブラリです．

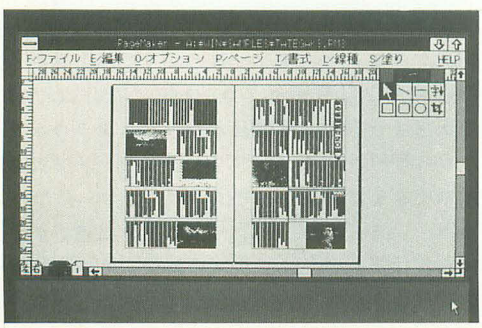
簡単にいえば**GKS**やPC9801の画面操作ライブラリのようなものであり，基本作画やポインティング・デバイス(マウスなど)の操作を行います．このインターフェースはきわめて直感的であり理解しやすいのですが，操作が原始的であるためアプリケーション側では細かな操作を指定しなければなりません．

Xウィンドウ Version 11になって，さらにその上のインターフェースとして**Toolkit**が用意されました．

INFORMATION

● パソコン用編集レイアウト・ソフトウェア

【商品名】アルダスページメーカー3.0J
【価格】148,000円（予価，年内発売予定）
【特徴】▶MS-Windows対応の全てのパソコン上で動作する
▶縦書/縦組，ふりがな，禁則処理などの日本語対応
▶文字と絵の切り貼り，デザイン修正，編集などが可能
◎Macで有名なページメーカーがようやくPC9801で実現できるようになった．ただ，Macに比べると画面は今一つ．
【問い合わせ先】(株)サムシンググッド ☎03-232-0801



● UNIXベースの低価格ワーク・ステーション

【型名】DEC station 2100
【価格】988,000円（ディスクなしのシステム）
【特徴】▶CPUはMIPS R2000（12.5 MHz）
▶メモリは8Mバイトから最大24Mバイト
▶DEC Windowsを標準でサポート
▶MS-DOSのエミュレーションをサポート(予定)
◎ソニーのNEWSも驚くほどの低価格．機能は最先端で言うことなし．売れてくれればよいがレインボーの例もあるし．
【問い合わせ先】日本DEC(株) ☎03-989-7111

Toolkit は，ウィンドウを **Widget**(仕掛とか道具という意味)というオブジェクトとして抽象化し，それに対する操作をメッセージ・パッシングで統一的に扱おうというものです．**Toolkit** の概念がなぜ大切かは，ウィンドウ・システムを使用したアプリケーション・プログラムを作成することを思い浮かべれば容易に納得できると思います．

我々がアプリケーション・プログラムを書こうとする場合，必要なウィンドウ・オブジェクトは必ず数種類のカテゴリに分類され，毎回同じようなウィンドウの作成を行うことになります．たとえば，スクロール・バー付きのメッセージ・ウィンドウとか，一行のコマンドを入力するための対話ボックスとかいったものがその代表ですが，だからといってそれが全く同じものではないというところが問題で，微妙な特徴付けやそれらの合成をどう取り扱うかが問題になってきます．**Toolkit** インターフェースは，その機能をオブジェクト指向の概念で統一的に取り扱えるようにしたものです．

Toolkit インターフェースがあっても実際に **Widget** を始めからユーザに作成させるのは無理があり，現実的にはあらかじめ標準的な **Widget** をライブラリとして用意しておき，それらを組み合わせて必要な **Widget** を合成するという方向で開発が進んでいます．

最近の X ウィンドウ Version 11 の配布テープには，M.I.T.の Project Athena で開発されている，いわゆる **Athena Widget** を始めとして各大学，メーカなどで開発されている **Widget** ライブラリが含まれており，それらを利用するときわめて短時間に複雑な画面構成を持ったアプリケーション・プログラムが作れます．もっとも，**X Toolkit Intrinsics** と **Widget** の仕様を理解していればの話ですが．

現在，公開されている **Toolkit** インターフェースは C，C++，Common Lisp などがありますが，普通に用いられる **Athena Widget** は C でインターフェースされるため，メッセージ送信の方法がいま一つ綺麗ではありません．やはりオブジェクト指向はそれにあった言語処理系でといいたいところですが，EWS が 99 % Unix あるいはその傍系を採用している現状では，実行速度をある水準に保つためには仕方のないことなのかもしれません．

X ウィンドウは，プログラマ・インターフェースとしての側面の他に，ユーザ・インターフェースとしての側面も持っています．X ウィンドウには当初から **xterm**(仮想端末ウィンドウ)，**xclock**(時計ウィンドウ)，**xload**(CPU 負荷メータ)，**xcalc**(電卓)などのアプリケーションがあり，それらとウィンドウ・マネージ

● 組み込み用 32 ビット・マイクロプロセッサ

【型名】i960CA (168 ピン PGA)
【価格】42,000 円(16 MHz)，58,000 円(33 MHz)/1000 個ロット単価
【特徴】▶ 1 クロックあたり 3 命令を実行する並列処理を採用
▶ 33 MHz クロックで 66 MIPS を実現する
▶ 4ch DMA，キャッシュ，割り込みコントローラ内蔵
▶ コプロセッサ 82596CA と直接接続可能
◎ 16 ビット CPU を組み込み用に使うことさえ大変なのに誰が使えるのかと心配になってしまうほどの処理能力をもつ．
【問い合わせ先】インテルジャパン(株) ☎ 029747-8511

● PC9801 用 EMS 拡張ボード

【型名】EMJmk II シリーズ
【価格】37,800 円(1 M)，64,800 円(2 M)，120,000 円(4 M)
【特徴】▶ 1 M バイトから最大 16 M バイトまで用意
▶ EMS 規格対応のソフト「MELWARE」を付属
▶ 386 のプロテクト・メモリを仮想 86 モードで使用可
▶ ハイレゾリューション・モードに対応
◎アプリケーションも大きくなり，EMS 対応の RAM ディスクは必需品に．それにしても開発競争がすさまじい．
【問い合わせ先】(株)メルコ ☎ 052-251-6891

ャの組み合わせでユーザ・インターフェースのカストマイズができます．

そのため，Xウィンドウの内部インターフェースを全く知らなくても，あらかじめXウィンドウを使用して作成されたアプリケーション・プログラムがいくつかあれば，Xウィンドウを使っている気分になれます．現実にはそういう使い方をするユーザが9割以上いるはずですから，最終的にXウィンドウがウィンドウ・システムの標準になるためにはXウィンドウの仕組みを知る必要のないユーザにどれだけの満足を与えられるかが重要なファクタになると思います．

● Xウィンドウの使いごこち

次に，日頃Xウィンドウを使用している側の印象を述べてみます．ユーザの立場でありがたいことは，複数ベンダ環境で操作環境が統一できる点にあります．

Xウィンドウを使っている限り，**xterm** はどのハードの上でも **xterm** であり，また，A社のマシン上で **xterm** を実行していながらサーバはB社のマシンということも普通に起こります．そのうえで時を刻む **xclock** はC社のマシンで動いているかもしれません．すなわち，Xウィンドウ自体がたいへんよくできた分散処理環境を自然に実現してくれます．各アプリケーション・プログラムのオプション引き数の形式が揃っているという点もユーザにとってありがたいことです．

一般に，Xウィンドウのアプリケーション・プログラムは起動時にウィンドウの大きさ，位置，色などを指定できます．これらの引数はそれを記述している **Widget** にそのまま渡されて **Toolkit** 側で処理されているため必然的に同じ形式になるわけですが，気分がよいものです．また，これはよいことといえるかどうかわかりませんが，一旦サーバを立ち上げてしまえばその機械の臭いが消えてしまうという点があります．

これは文字フォントまでXウィンドウが提供するためでもありますが，そのためか，国産メーカのXウィンドウの中にはそれを避けるために差別化を図っているものがあるらしく，機能が追加されている場合があります．特に，日本語回りに多いのですが，そのためにある機械の上の **kterm** にあるオプションが他社ではなかったりすることが多々あります．Xウィンドウのポリシではソースが付いてくるはずですから，それを見ようと思ってもなぜか国産メーカのXウィンドウにはソースが付いてきません．

Xウィンドウが使用できるハードウェアのレンジも広く，現時点でアナウンスのあるサーバは下位はIBM PCクラスまで広がっており，386ラップトップPCでも正式にサポートするものが現れています．大学のようにどうしても複数ベンダ環境になってしまうところではサポート・マシンのレンジの広さは重要です．もっとも，それによってサーバの能力差が拡大するのですが，それを吸収するための努力が各所になされています．

たとえば，カラー・サポートでは，Xウィンドウの習慣では色は英文の色名，たとえば“Turquoise”で指定したり，RGBの比率で指定したりしますが，それは絶対色ではなくサーバが近似して自分が表現できる範囲で近似します．したがって，その色はないと冷たくエラー・メッセージで停止することはありません．

ユーザ・インターフェースにとって大きなウェイトを占めるのが速度です．実際に，Xウィンドウの速度は速いのでしょうか．この評価は難しいのです．一つはXサーバのデバイス・ドライバの作成は機器メーカあるいはユーザに任されているため，それによって速度が大幅に変わってしまうからです．筆者がよく使う米国S社のカラー・モニタの上のXウィンドウはカラー・サポートのXウィンドウを立ち上げると信じられないくらい遅くなります．速いXサーバをサポートしないのは自社のウィンドウを売るためではないかと勘ぐりたくなるほどです．

しかし，普通に使うモノクロ・ビット・マップのXウィンドウは十分実用になる速度で，安定性も十分です．Xウィンドウの速度はビット・マップの速度もさることながらTCP/IPの速度もネックとなるはずです．

現状ではTCP/IPのスループットは200～300Kバイト/secが上限ですから全画面のビット・マップ転送を行うと当然ひと休みします．Xウィンドウ側でも不必要なビット・マップ転送を抑えるためにサーバ側にビット・マップを蓄えるプロトコルを用意し，レスポンスを低下させない努力をしていますから，これをアプリケーションがうまく使えば当面必要な速度は保てるようです．

Xウィンドウの前身はWという名前のシステムだったと聞いています．Xウィンドウ Version 11 のプロトコルは今後変更しないということになっているそうですから，次に出て来る版は Version 12 ではなくYになるのでしょうか．

〈山本　強〉

(＊1) TCP/IPは，米国国防総省の一部局であるDARPAが提案したコンピュータ通信回線用のプロトコル．DARPAが援助したARPANETに採用されたことから広く普及した．OSI参照モデルでいえば，ネットワーク層とトランスポート層に相当する．

特集　1989年11月号

MS-DOSとデバイス・ドライバ

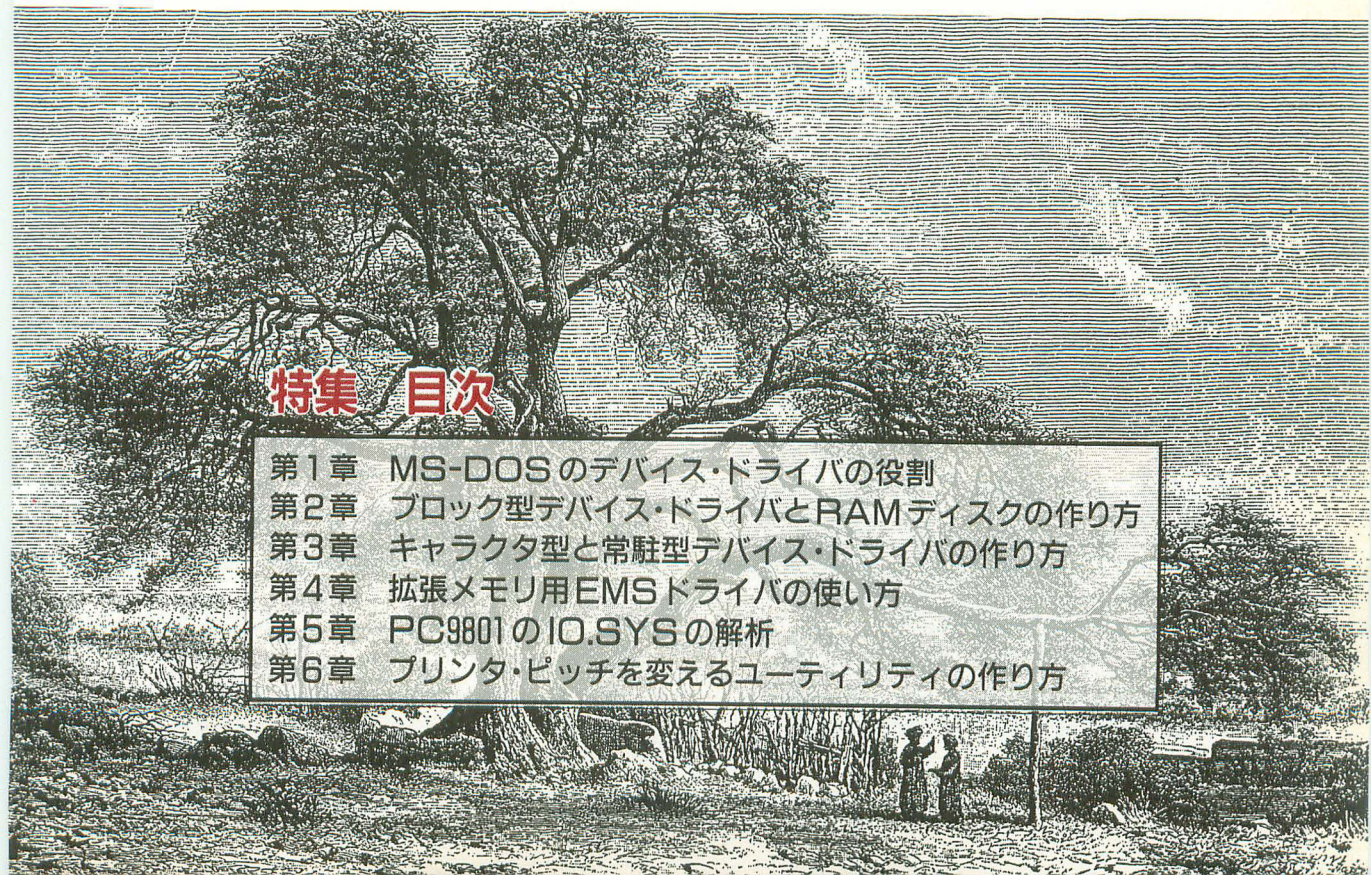
過去の技術では実現できなかった新しい周辺装置が登場しても，従来のソフトウェアを変えないでコンピュータの本体に簡単に接続できたら，どんなに便利でしょう．MS-DOSでは，様々な周辺装置をコントロールするために，本体のシステム・プログラムと周辺装置に対応するプログラムが分離しています．そして，新たに周辺装置が必要となれば，新規のデバイス・ドライバとして追加するだけでよいのです．

といっても簡単にそのデバイス・ドライバが作れるわけではありません．本特集では，やさしい事例をたくさん示しながら，デバイス・ドライバのプログラミング方法を学んでいきます．

第1章では，デバイス・ドライバの必要性とその概要を説明します．つづいて第2章では，ブロック型のデバイス・ドライバの事例としてRAMディスクを製作し，そのプログラムを作ります．そして，第3章では，キャラクタ型の例とTSRと呼ばれる常駐型デバイス・ドライバを紹介します．

第4章では，主記憶の容量の問題から拡張メモリを使うために利用の進んでいるEMS規格を使うためのプログラミングを示します．

第5章では，PC9801のプリンタのドライバを解析します．2バイト・コードを扱う漢字処理は少々複雑です．そして，第6章では，漢字とANK文字の間隔の比率を1：2のままでパイカ以外の文字が出せるプログラムの解説を行います．



特集　目次

第1章　MS-DOSのデバイス・ドライバの役割
第2章　ブロック型デバイス・ドライバとRAMディスクの作り方
第3章　キャラクタ型と常駐型デバイス・ドライバの作り方
第4章　拡張メモリ用EMSドライバの使い方
第5章　PC9801のIO.SYSの解析
第6章　プリンタ・ピッチを変えるユーティリティの作り方

第1章 簡単に各種の周辺装置を接続することができる理由

MS-DOSのデバイス・ドライバの役割

パソコンの能力の向上にはめざましいものがあります．周辺装置として，パソコンに接続できるものが数多く開発されています．MS-DOSではデバイス・ドライバというシステムによって，それらの装置をパソコン本体に追加することが容易にできます．

井芹 陽一

パーソナル・コンピュータが登場してから，早くも10年以上の歳月が流れました．この間に，半導体技術はめざましい進歩を遂げ，8ビットCPU，16ビットCPU，32ビットCPUまで，知らず知らずの間に使ってきた感があります．

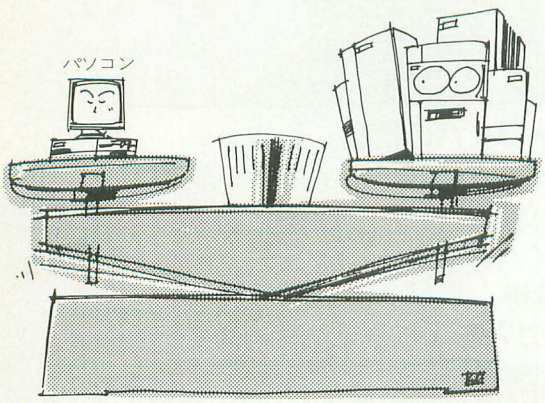
それは，その時々のCPUの能力に応じたOSやソフトウェアの力によるところが大です（筆者の個人的な感じでは，ソフトのほうが後手後手にまわっている気がしないでもないが）．

さて，パソコンといえばMS-DOS，MS-DOSといえばパソコンというくらい，パソコンのOSとしてMS-DOSは不動の地位を占めています．本稿では，このOSの大きな特徴の一つであり，その存在に大きな影響を及ぼしているデバイス・ドライバについて簡単に説明します．

デバイス・ドライバと聞くと，何やら非常に難しいものと感じがちですが，けっしてそのようなことはなく，一般に作成するソフトと何ら変わるところはありません．所ジョージ氏のキャラクタのように“かるーくかるーく”取り組んでいただきたいと思います．

なお，本稿ではハードウェアとして，PC9801シリーズをパソコンの代表選手としています．また98か，と思われる向きもあるかと存じますが，ご容赦願いたいと思います．また，MS-DOSはVersion3以降を前提としています．

〈図1〉最新のパソコンの処理能力は高い



パソコンも一人前のコンピュータ・システム

さて，現在のパソコンに目を向けてみると，CPUはインテル社の80286や80386，OSはMS-DOSが主流となっています．

これらCPU単体の性能は，総合性能では比較の対象になりませんが，1970年代前半の大型汎用機とある部分で拮抗しています（図1）．また，周辺装置に目を向けてみると，ハード・ディスク装置は40Mバイト以上は当たり前，中には数100Mバイトというものもあります．さらに，高解像度のCRTディスプレイ，X-Yプロッタ，レーザ・プリンタ，光ディスク装置，補助記憶装置としてのRAM，LANアダプタ，マウス，イメージ・スキャナなどと，数え上げれば切りのないくらいのものがパソコンで利用されています（図2）．

各々の性能は別として，システム的には大型汎用機と何ら変わるところはありません．

パーソナル・ユースでこのようなシステムを利用できる現在と，筆者が初めてコンピュータに接した頃をくらべると隔世の感があります．

ハードウェア制御の必要性 ― BIOSの存在

パソコンに内蔵された周辺装置（フロッピ・ディスク，ハード・ディスク，CRTコントローラ，プリンタ，マウス，シリアル通信回線etc.）や外部に接続された周辺装置を利用する場合，ハードウェアによる制御だけでなく，ソフトウェアによる柔軟かつ，木目細かな制御があって始めてその性能を十分に引き出すことができます．したがって，その制御ソフトのできいかんによってシステムの能力が決定されるといっても過言ではありません．

もし，そのようなソフトウェアをユーザが自分自身で作らなければならないとしたらどうでしょうか．機械制御のソフトウェアを少しでも作ったことのある方ならおわかりと思いますが，人それぞれ作り方も違い

ますし，その繁雑さは言葉で言い表せないものがあります．

● **互換性なんて**

当然，互換性のあるソフトウェアなどは作られるはずもありませんので，パソコンは一部マニアのオモチャでしかなくなり，現在のように，市販ソフトが数多く流通することはなかったでしょう．

そこで，ハードウェア・メーカは，内蔵しているハードウェア(またはインターフェース)に関して，その基本制御ソフト(BIOS：Basic Input Output System)をハードウェアに付属したかたちで提供しています．これは，パソコンに内蔵されているシステムROM(Read Only Memory)やフロッピ・ディスクに書き込まれています．

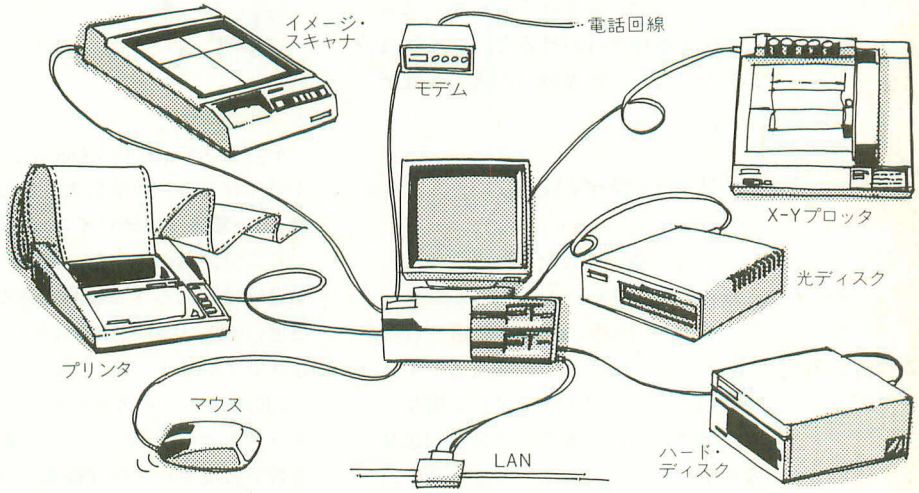
これで，ハードウェアをアクセスする手順が統一され，また，パソコン・ユーザは繁雑なBIOSの作成作業から開放されます(**図3**)．

ここに，メーカとユーザ両者にとって，BIOSの存在意義があるのです．

OSとその管理下にあるハードウェア
―資源大国パソコン・システム―

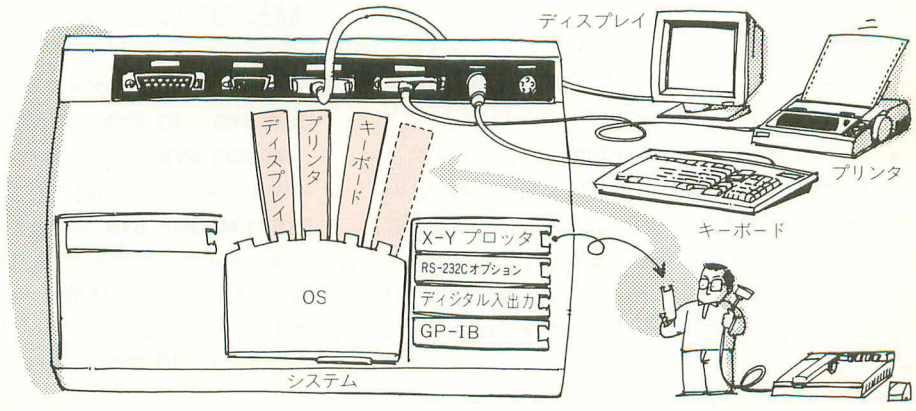
OSの目的を教科書的に書くと“資源の有効利用を図り，人を含めたコンピュータ・システムを統括管理する”となります．ちょっと難しい表現なのですが，

〈図2〉
パソコンにつながらない周辺装置はない？



〈図3〉
BIOSがソフトとハードの仲立ちをする

〈図4〉
新しい周辺装置には新しいデバイス・ドライバが必要



資源とはメモリ，ハード・ディスク，フロッピ・ディスク，そのほかの周辺装置のことです（図4）．

日本は資源小国ですが，パソコンは資源大国です．いろいろな資源をもっています．しかし，資源はただもっているだけでは何の役にもたちません．有効利用してこそ役にたつというものです．

MS-DOSは，私達の気が付かないところで この資源管理を上手（ゴマスリです．本当はへたくそ）に行っています．MS-DOSからアクセスできる周辺装置はすべてその管理下にあります．汎用機のそれと比べるとまだまだ幼稚なOSですが，PC9801シリーズに代表されるパソコンにはちょうど良いものです．

なぜなら，DMAを含めたバス転送能力が非常に弱いため，ハード・ディスク装置などの補助記憶装置との間でロールイン・ロールアウトを行うような，マルチジョブやマルチタスク処理を行わせると，バス転送のオーバヘッドが大きくなり，処理能力が著しく低下してしまうからです．

デバイス・ドライバとは

前に，資源という言葉を使いましたが，ここでは，資源＝デバイスと考えてください．したがって，デバイス・ドライバとは，“MS-DOSの管理下に置かれた周辺装置をアクセスするソフトウェア”のことです．

それでは，なぜこのようなソフトウェアが必要なのでしょうか．読者の中で，現在市販されている周辺装置の種類をすべて列挙できる方は少ないと思います．まして，将来の周辺装置まで予測することはまず無理でしょう．メーカ・サイドの事情もまったく同じことです．

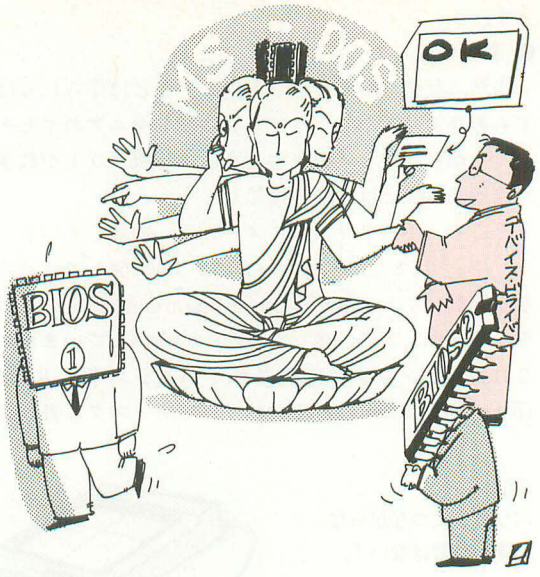
● 将来に出現する周辺装置に対応するために

ところが，パソコンはパーソナル・コンピュータの名が示すとおり，パーソナル・ユースに使われることが多く，そこで使われる周辺装置も千差万別です．コンピュータ・メーカが予め，そのようなものを制御するソフトウェアを作っておくことなど土台無理な話です．

したがって，周辺装置を制御するソフトウェアを自由にOSに組み込むことができたら，どれだけ便利で拡張性に富んでいるか知れません．その目的を達成するのがデバイス・ドライバです．

オフィス・コンピュータや汎用機では，デバイス・ドライバ類をOSに登録する作業は大変なもので，パソコンのように簡単にくっつけるだけではできません．システムの再構築を行わなくてはならず，オフコン・クラスでも数時間かかります．このような作業は，ソフトによって動作環境が異なるパソコンではとてもできません．

〈図5〉デバイス・ドライバはBIOSと共に働く



その意味で，簡単にOSに登録できるデバイス・ドライバの存在は，非常に大きなものといえます．

● 広い意味ではデバイス・ドライバもBIOSといえる

ここで読者の中には，BIOSとデバイス・ドライバとの違いは何だろうと疑問を持たれる方もいらっしゃると思います（広い意味ではデバイス・ドライバもBIOSなのですが）．

BIOSはデバイスそれぞれに作られた，ハード寄りのソフトウェア（デバイスを選択するものとして，16進数で表現される物理機番が使われる）で，直接MS-DOSに登録することはできません（図5）．

それに対して，デバイス・ドライバは，BIOSよりももっとBasicやdBASEのようなアプリケーション・ソフト寄り（デバイスを選択するものとして，記号や略名が使われる．例えば**A:**，**PRN**，**AUX**など）で，デバイスがより論理的に取り扱われています．また，MS-DOSに直接登録することができます．

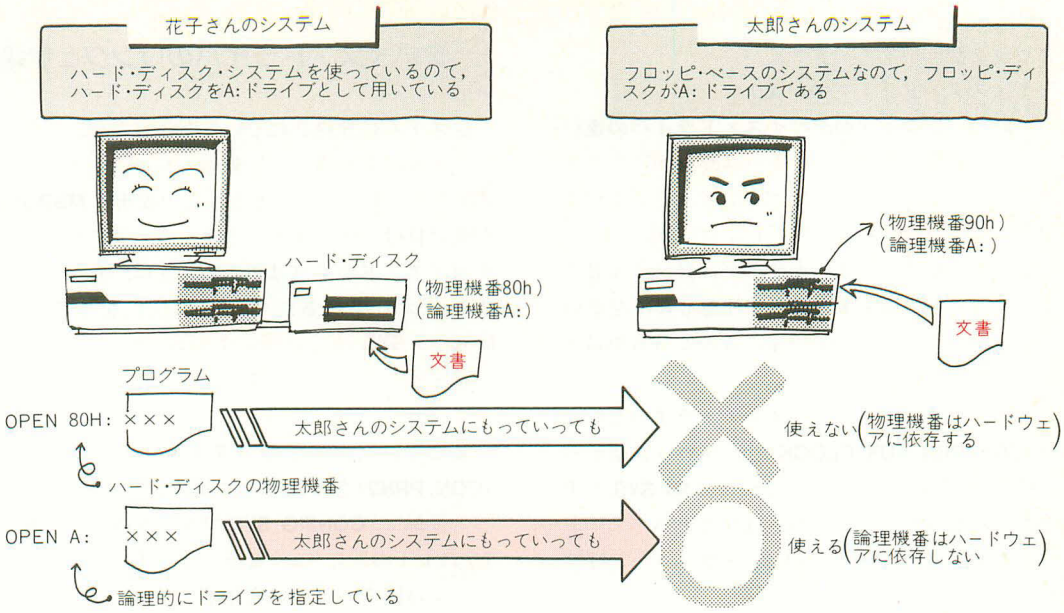
デバイス・ドライバは，ユーザが自由にMS-DOSに登録することができるのです．

MS-DOSとデバイス・ドライバの関係

オペレーティング・システムMS-DOSは，**MSDOS.SYS**と**IO.SYS**とから構成されています．

MSDOS.SYSは本来のオペレーティング・システムの機能を果たします（詳しくは参考文献を参照）．**IO.SYS**は**MSDOS.SYS**が使用する入出力装置用のコントロール・プログラム（つまりデバイス・ドライバ）で構成されています．機種やバージョンによって異なりますが，基本的に次のデバイス・ドライバが予めメーカによって，**IO.SYS**に用意されています．

〈図6〉論理的扱いをすることで汎用性が生まれる



▶**CON**……コンソール入出力．
キーボードよりデータを入力したり，ディスプレイに出力したりする．

▶**AUX**……補助入出力．
RS-232C インターフェースなどとの間で入出力動作を行う．

▶**PRN**……プリンタ出力．
プリンタにデータを出力する．

▶**CLOCK**……リアルタイム・クロック．
時計機能．

▶**A:～Z:**……補助記憶装置入出力．
フロッピ・ディスクやハード・ディスク装置との間で入出力動作を行う．

ここで，デバイスにそれぞれ名前が付いているのは，デバイスを論理的に扱うためです．

● **個有のデバイス名をつけて論理的に扱う**

例えば，太郎さんのシステムにはフロッピ・ディスクしかないが，花子さんのシステムはハード・ディスクもある場合，もしデバイスを論理的(**A:**とか **B:**とか)に扱うことができなかったら，物理機番をそれぞれで使うプログラムの中にセットしなければならず機種間でプログラムの互換性がなくなります(図6)．

また，MS-DOS ではこの論理デバイスをファイルと同様に扱っています(ディスクは別)．したがって，システムに登録されている(**IO.SYS** に登録されているものと，ユーザが登録したもの)デバイス名と同じ名前のファイルをディスク上に作ることはできません．また，同じ理由でビルトイン・コマンドの **COPY** 命令などで，ディスク・ファイルと同様にデバイス名が使用できます．

実は，オペレーティング・システム(**MSDOS.SYS**)でさえ，論理デバイス(**CON, PRN** など)が物理的にどのデバイスに接続されているのか知らないのです．知っているのはデバイス・ドライバだけです．

つまり，OS(デバイス・ドライバ以外)を会社の社長に例えると，デバイス・ドライバは部課長ということになります．

デバイス・ドライバの種類

MS-DOS の大きな特徴の一つにデバイス・ドライバを自由に増設できると，先に書きましたが，これは，ユーザそれぞれによって異なるシステムを構築するうえで非常に有効で，必要最小限の投資で目的を達成できます．

とくに，日本で使う場合，日本語変換機能が必須ですが，MS-DOS の場合，各種の日本語仮名漢字変換フロント・エンド・プロセッサ(FEP)を自由にデバイス・ドライバとして登録できますので，実に環境の良いシステムをつくることができます．

この場合のデバイスは，人と考えるとよいでしょう(人もシステムの構成要素)．

さて，デバイス・ドライバは，大きく次の二つに分けることができます(例外として，何もしない **NULL** ドライバがありますが，これは **MSDOS.SYS** の中で特別に扱われる)．

(1) キャラクタ型デバイス・ドライバ
　　コンソール(**CON**)
　　プリンタ(**PRN**)
　　補助入出力(**AUX**)

時計機能　(**CLOCK**)

etc.

(2) ブロック型デバイス・ドライバ

ディスク・ドライブ(**A:**～**Z:**)

● **キャラクタ/ブロック型デバイス・ドライバの違い**

キャラクタ型デバイス・ドライバは基本的にデータを1文字ずつ扱い，ブロック型デバイス・ドライバはデータをブロック単位(1024バイトなど)で扱います．また，ブロック型デバイス・ドライバはファイル名でなく，ドライブ名で識別することに注意してください．

いかなるデバイス・ドライバも，このどちらかに当てはまります．

先の例で，キャラクタ型デバイス・ドライバとして4種類(**CON, PRN, AUX, CLOCK**)と，ブロック型デバイス・ドライバをあげましたが，これらは**IO.SYS**の中に標準的に用意されているため，ほかのデバイス・ドライバと区別して，“標準デバイス・ドライバ”と呼びます．

ブロック型デバイス・ドライバは，**A:**から**Z:**までの名称しか使用できませんが，キャラクタ型デバイス・ドライバはファイル名として許される名称は何を使ってもかまいません(ただし，8文字までで，ディスク・ファイルとして登録されているものは使用できない)．**AUX2**や**PRN2**などの名前が自由に使えるのです．

このようにして，デバイス・ドライバとしてシステムに登録しておくと，ビルトイン・コマンドのCOPY命令などで，ファイルとして取り扱うことができます(図7)．

デバイス・ドライバのリンクと登録

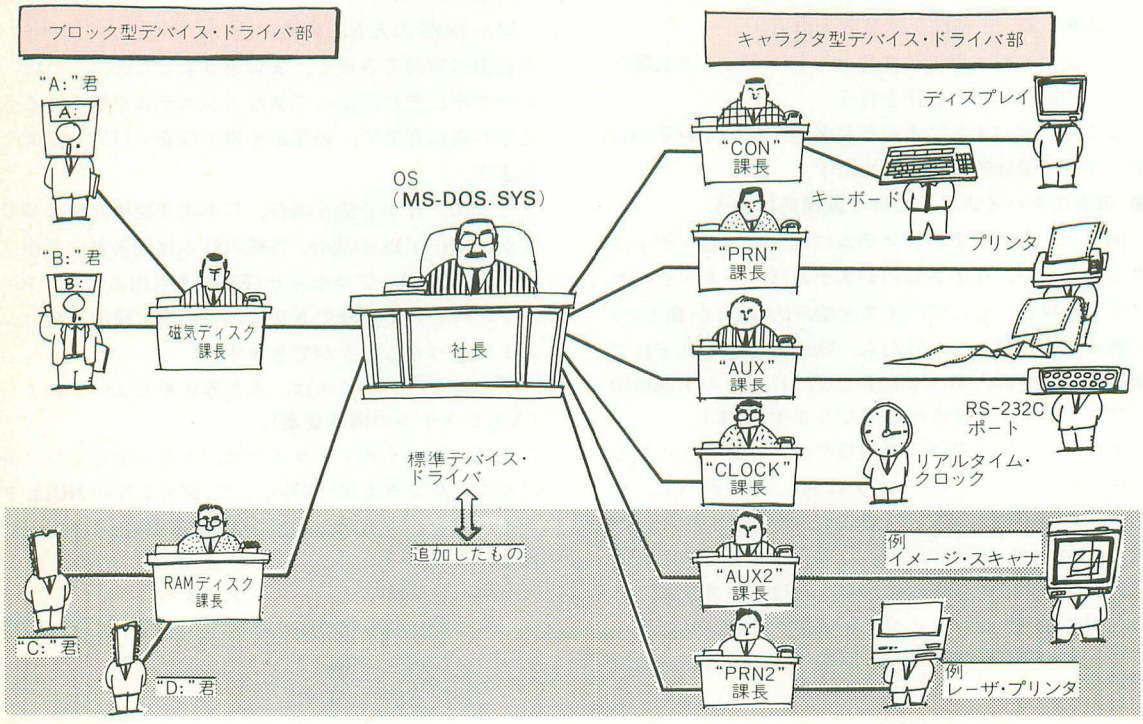
システムに登録されている周辺装置をプログラムよりアクセスする場合，一般に，システム・コール**INT 21h**を使います．このとき，この要求を**MSDOS.SYS**が受け付け，デバイス・ドライバをアクセスします．デバイス・ドライバは図8のように，あたかも一本のチェーンで接続されているようになっています．**MSDOS.SYS**はこの図の上側(**CON**)より順にチェーンを手繰ってアクセス要求の出ているデバイスを捜します．

ここで，二つのキャラクタ・デバイス・ドライバ(**CON, PRN2**)をシステムに登録してみます．システムへの登録は，**CONFIG.SYS**ファイルの中に定義します(**リスト1**参照)．このとき，デバイス・ドライバのチェーンは図9のように変化します．

ここで注意することは，もともと用意されている標準の**CON**デバイスは無効になることです．同一名のキャラクタ型デバイス・ドライバが存在するときは，**CONFIG.SYS**ファイルの中で最後に定義したデバイス・ドライバのみが有効となります．この理由は，**MSDOS.SYS**がチェーンの上側から順にデバイスを捜すことから明白です．

ブロック型デバイス・ドライバの場合は，チェーンの中での位置関係はキャラクタ型デバイス・ドライバ

〈図7〉ブロック型とキャラクタ型のデバイス・ドライバの追加



と同じなのですが，ドライブ名はすでに登録されているドライブ名の次から割り付けられます(図10)．

デバイス・ドライバの構造

デバイス・ドライバは，一本のチェーンで接続されているとお話ししましたが，そのためにデバイス・ドライバは特別な構造になっています(図11(a))．この図のデバイス・ヘッダについて説明します．この言葉のニュアンスからすると，何やらとてつもなく難しいもののような気がしますが，実はなんてことのないもので，たんなる“履歴書”程度に考えてください．

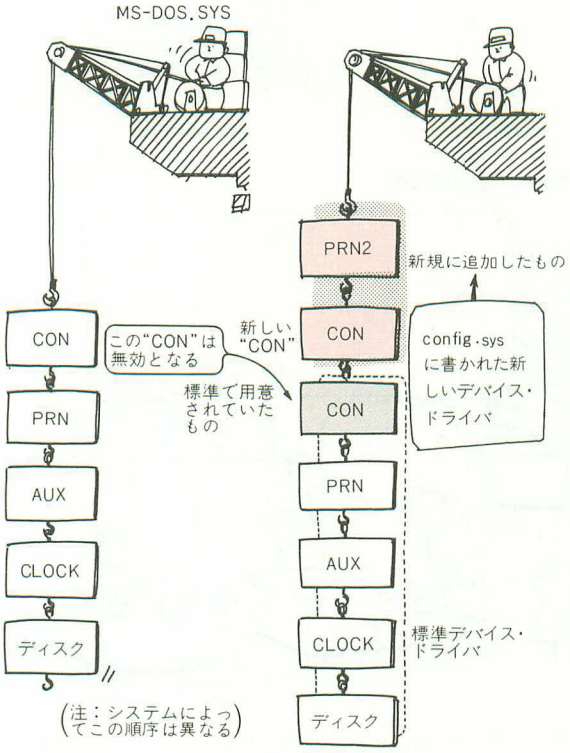
図11(b)のように，“デバイス・ドライバ君”が“MS-DOS社”に面接試験に行ったとしましょう．このとき，デバイス・ドライバ君は履歴書(デバイス・ヘッダ)を持参しています．さっそく社長(MSDOS.SYS氏)直々の面接試験が行われ，めでたくMS-DOS社に入社の運びとなります．履歴書に示された内容から，社長はデバイス・ドライバ君を適材適所に配置します(デバイス・ドライバの登録)．

以上のような経緯で，デバイス・ドライバがMS-DOSに登録されますが，そのときに必要な情報(履歴)がデバイス・ヘッダです．

デバイス・ヘッダは18バイトからなる固定領域で，デバイス・ドライバには絶対に必要です．デバイス・ヘッダの内容について詳しく見てみましょう．

〈図8〉デバイス・ドライバはつながっている

〈図9〉新しいデバイス・ドライバはトップにつなげられる

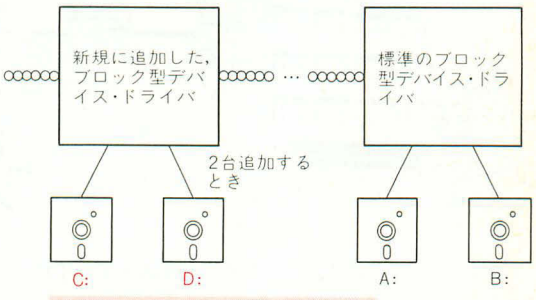


● リンク・ポインタ

―― 前に登録したデバイス・ドライバ(登録してないときはIO.SYS内の最初のデバイス・ドライバ)とつなぐ鎖の役目をする ――

リンク・ポインタは2ワード(4バイト)で構成され，次のデバイス・ドライバの先頭のアドレスを指します．

〈図10〉ブロック型デバイス・ドライバを追加するとディスクの番号が増える



〈リスト1〉新規のデバイス・ドライバの追加はCONFIG.SYSに記述を追加するだけ

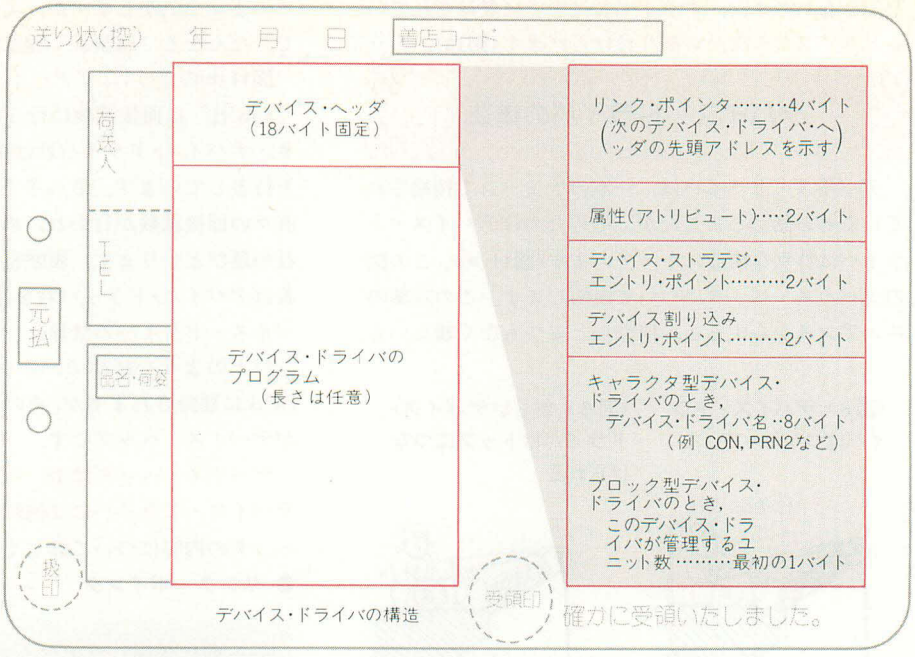
デバイス・ドライバの登録

CONFIG.SYSファイルの中に次の例のように登録する

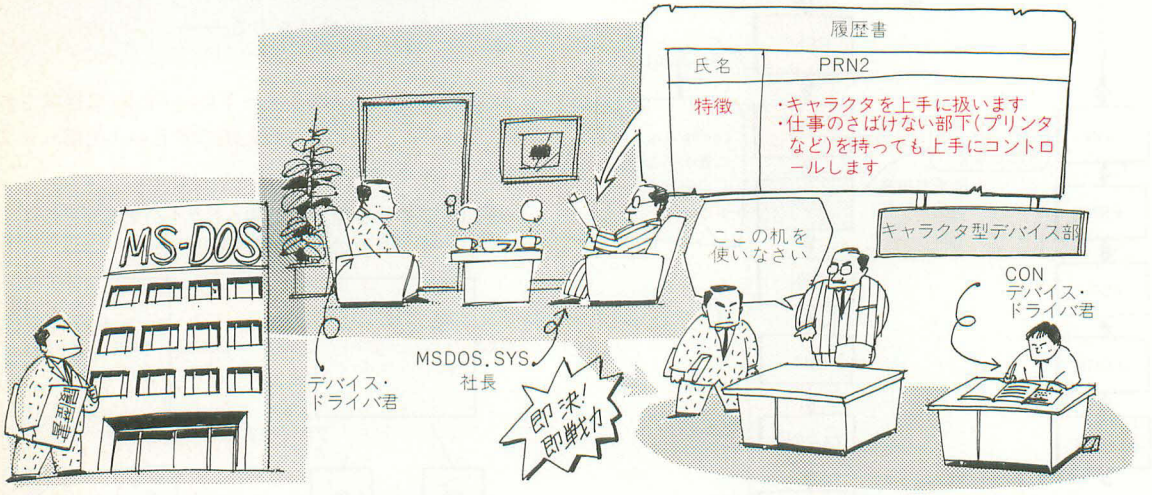
```
DEVICE=CONSOL.DRV ――► デバイス・ドライバ“CON”のファイル名(デバイス・ドライバ名とは異なる)
DEVICE=PRINT2.DRV ――► デバイス・ドライバ“PRN2”のファイル名(デバイス・ドライバ名とは異なる)
                       (デバイス・ドライバ名の定義は後述)
```

この例では，OSの起動時に“CON”，“PRN2”の順に登録される

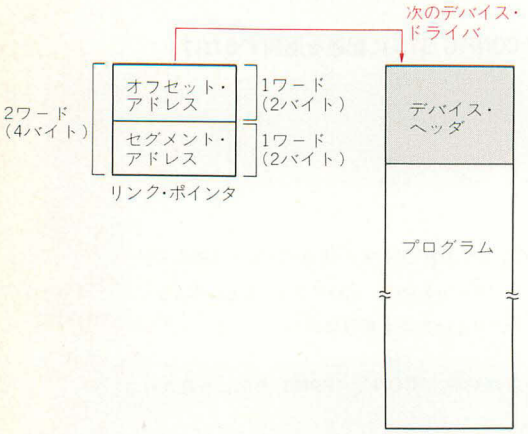
〈図11(a)〉
デバイス・ドライバの
構造とヘッダの内容



〈図11(b)〉 デバイス・ドライバの登録の手順



〈図12〉 リンク・ポインタで次のデバイス・ドライバとつながっている



リンク・ポインタの最初の1ワードはオフセット・アドレスで，次の1ワードはセグメント・アドレスです(図12).

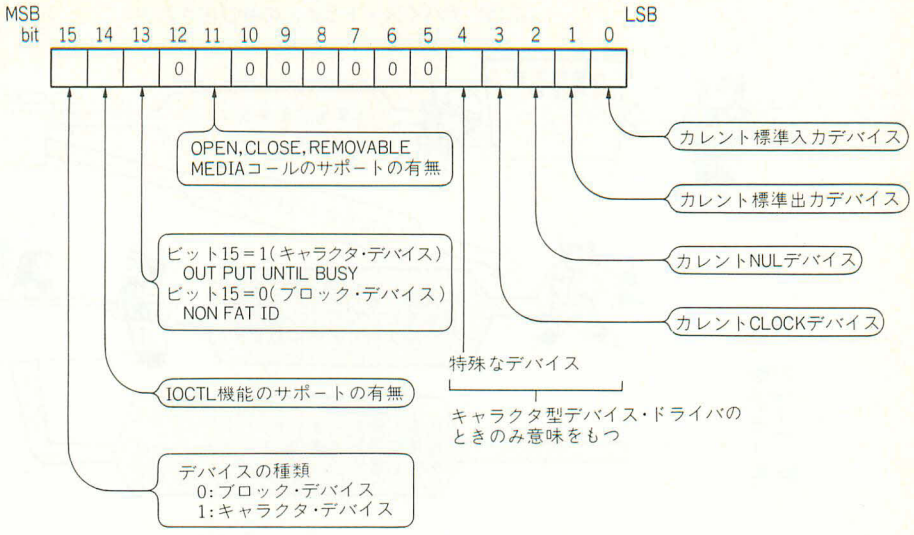
通常，デバイス・ドライバをシステムに組み込むときは，ここに−1(FFhを4バイト)を書いておきます．そうすると，組み込み時に自動的に先に登録されているデバイス・ドライバの先頭アドレス(デバイス・ヘッダの先頭アドレス)がセットされます．

● 属性(アトリビュート)フィールド

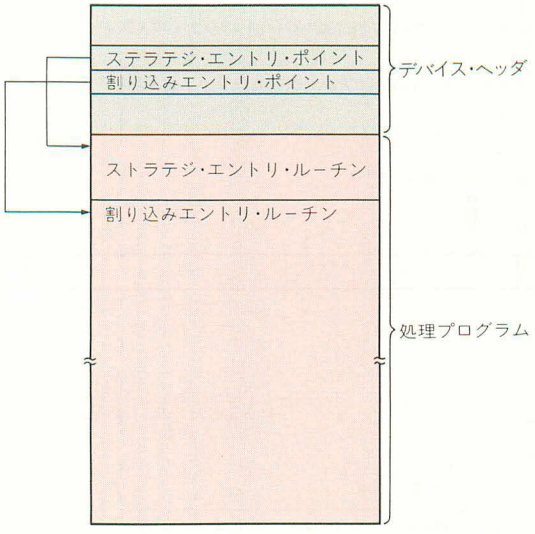
― 履歴書の履歴にあたる ―

このフィールドは1ワード(2バイト)で構成され，それぞれのビットが(図13)のような意味をもっています．第2章以降で詳しく説明されています．

〈図13〉
デバイス・ドライバのヘッダ内のアトリビュートの内容



〈図14〉エントリ・ポインタは各処理の入口を示す



● デバイス・ストラテジのエントリ・ポイント

― 作業依頼を受け付けるときの窓口 ―

デバイス・ストラテジのエントリ・ルーチン(詳細は後述)の開始アドレスをセットします(図14).

● デバイス割り込みのエントリ・ポイント

― 作業命令を受け付けるときの窓口 ―

デバイス割り込みエントリ・ルーチン(詳細は後述)の開始アドレスをセットします. デバイス・ストラテジのエントリ・ポイントやデバイス割り込みのエントリ・ポイントは, 共に1ワード(2バイト)しかありませんので, これらの開始アドレスはデバイス・ヘッダと同一のセグメントに存在しなくてはなりません.

● デバイス・ドライバ名(またはユニット数)

― 履歴書の中の氏名にあたる ―

キャラクタ型デバイス・ドライバの名前をセットします.

ブロック型デバイス・ドライバの場合は, 管理しているユニット数を最初の1バイトにセットします.

次に, 図14の中のプログラム領域ですが, これは大きく分けて二つになります. 一つはストラテジ・エントリ・ルーチンと, もうひとつは割り込みエントリ・ルーチンです.

ストラテジ・エントリ・ルーチンは, MS-DOSからデバイス・ドライバのコールを行うとき, 処理に必要なデータ(リクエスト・ヘッダ)をデバイス・ドライバに取り込む処理を行います.

割り込みエントリ・ルーチンは, ファンクション(17種類)に応じて, 実際に処理を行うもので, ここがデバイス・ドライバの本体といえます.

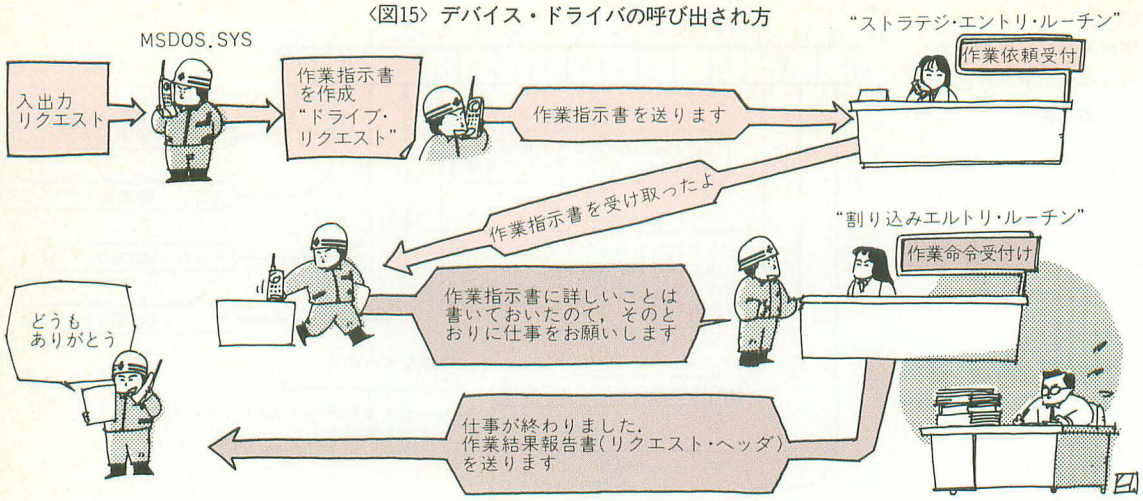
デバイス・ドライバの呼び出し
― デバイス・リクエスト ―

デバイス・ドライバの大まかな構造がご理解いただけたでしょうか. さて, 次にMS-DOSがデバイス・リクエストを行う場合, どのような手順で行うかを説明します.

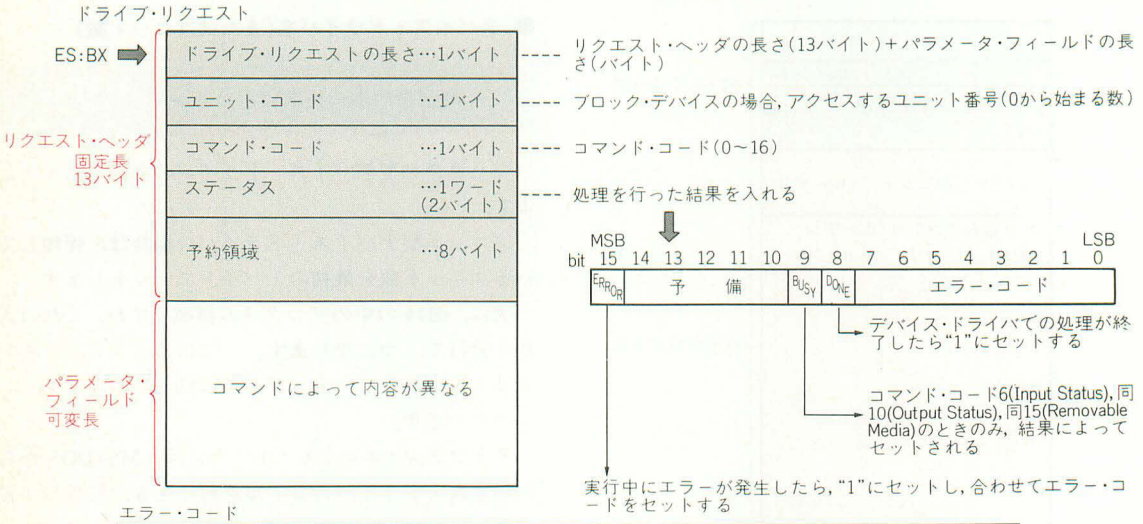
図15にその手順を示します. ここで注意することは, ある作業をデバイス・ドライバに行わせる場合, 2回もデバイス・ドライバの内部をコールしていることです(ストラテジ・エントリ・ルーチンと割り込みエントリ・ルーチン).

1回目のコール(ストラテジ・エントリ・ルーチン・

〈図15〉デバイス・ドライバの呼び出され方



〈図16〉ストラテジ・エントリ・ルーチンによって引き渡されるパラメータ



エラー・コード

00h ライト・プロテクトされているディスクに書き込もうとした	07h メディア・タイプが間違っている
01h 指定されたユニット番号は存在しない	08h セクタが見つからない
02h ドライブの準備ができていない	09h プリンタの用紙切れ
03h 無効なコマンド（コマンド・コードが間違っている）	0Ah ライト・エラー
04h CRCエラー	0Bh リード・エラー
05h ドライブ・リクエストの長さが間違っている	0Ch その他のエラー
06h シーク・エラー	0Fh 不正にディスク交換された

コール）では，デバイス・ドライバがコールされたときに必要な各種のパラメータ（作業指示書）を，MS-DOSからデバイス・ドライバに引き渡します．これは，図16のようにリクエスト・ヘッダと呼ばれる13バイト固定の領域と，パラメータ・フィールド（各コマンドによって内容が異なる）から構成されます（ここでは，この二つを合わせてドライブ・リクエストと呼ぶことにする）．

MS-DOSは，ドライブ・リクエストの最初のアドレスを，ESとBXレジスタにセットしてストラテジ・エントリ・ルーチンをコールします．

2回目のコール（割り込みエントリ・ルーチン・コール）では，MS-DOSが望む処理を，ストラテジ・エントリ・ルーチンによって受け取ったパラメータにしたがって行います．

一見面倒なことを行っているようですが，意味深長なものがあります．すなわち，将来MS-DOSがマルチタスク化したとき，プログラムの手直しが少なくてすむようにしているようです．

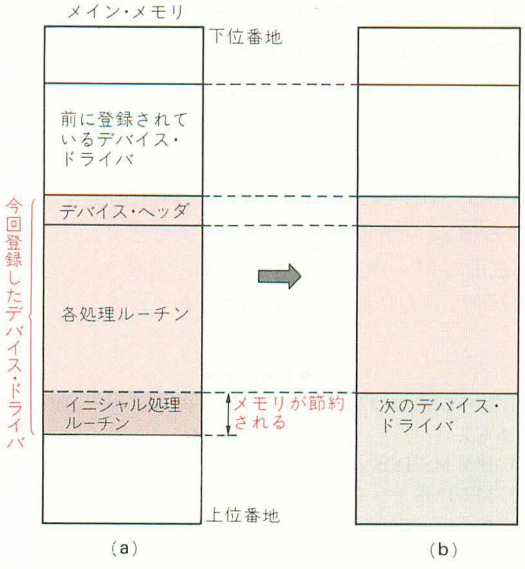
◆ ストラテジ・エントリ・ルーチン

このルーチンは，デバイス・ドライバが本来の仕事をする前にMS-DOSによってコールされます．ここでの目的は，MS-DOSよりドライブ・リクエスト（ES：BXで示される番地からセットされている）を受け取り，その開始番地（ESとBX）をデバイス・ドラ

〈リスト2〉
ストラテジ・エントリ・ルーチンの記述例
(MASMで記述)

```
strategy        proc    far
                mov     word ptr cs:req_head_ofs, bx ……オフセット・アドレスを退避する
                mov     word ptr cs:req_head_seg, es ……セグメント・アドレスを退避する
                ret     ……Far Returnとなる
strategy        endp
req_head_ofs :
                dw      0 ……オフセット・アドレスの退避領域
req_head_seg :
                dw      0 ……セグメント・アドレスの退避領域
```

〈図17〉イニシャル処理部を切り離せばメモリが増える



イバ内の作業領域に退避することです．退避した開始番地は割り込みエントリ・ルーチンで使われます．

リスト2に，アセンブリ言語によるプログラムの一例を示します．このとき，MS-DOSからはFAR CALLされることに注意してください．

◆ 割り込みエントリ・ルーチン

MS-DOSは，ストラテジ・エントリ・ルーチンをコールした後，次に割り込みエントリ・ルーチンをコールします．実は，この処理ルーチンがデバイス・ドライバの本体で，とても重要です．

図15のように，デバイス・ドライバはストラテジ・エントリ・ルーチンによって，“作業指示書”ともいえるドライブ・リクエストを受け取り，それに従って作業を行います．そして，作業が終了すれば，“作業結果報告”を行いますが，これもリクエスト・ヘッダに結果を書いてMS-DOSに報告します．

ここでリクエスト・ヘッダについて少し追加説明をしておきます．

まず，ユニット・コードとは，ブロック型デバイス・ドライバのときのみ意味をもち，例えば，一つのブロック型デバイス・ドライバで三つのユニットを管理する場合，今からアクセスするのはどのユニットかを示します(0から始まる番号で表す)．

MS-DOSが作業指示を行う場合，何処かの誰かさん(?)のように，だらだらと作業指示をしたりしません．簡潔明瞭に行います．そのために，作業の種類を17種に分類しています．MS-DOSはそれをコマンド・コード(0～16)で指示します．次にコマンド・コードとその機能の一覧を示します．

コマンド・コード	機能
0	初期化
1	メディア交換の有無をチェック
2	BPBの作成
3	リード(IOCTL)
4	リード
5	リード(連続非破壊)
6	リード・ステータスの取得
7	リード・フラッシュ
8	ライト
9	ライト(ベリファイ付き)
10	ライト・ステータスの取得
11	ライト・フラッシュ
12	ライト(IOCTL)
13	デバイスのオープン
14	デバイスのクローズ
15	メディアの交換の可，不可の取得
16	ライト(busyでも復帰可)

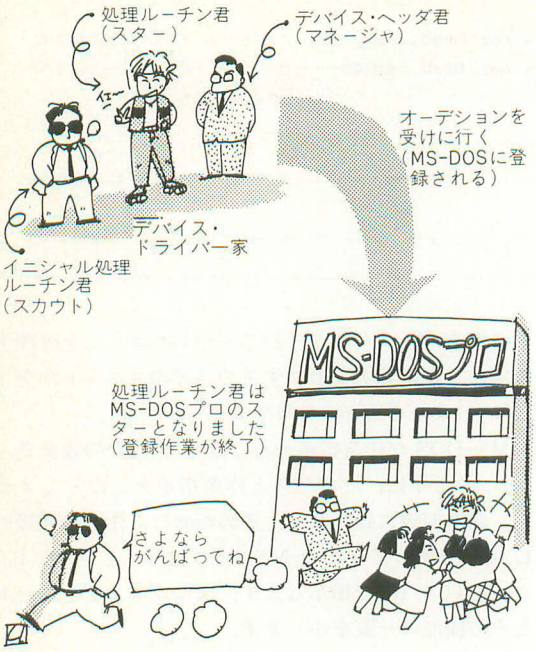
各機能についての説明は，第2章以降で詳しく説明されますので，ここでは省略します．ただし，コマンド・コード0(イニシャル処理)については，デバイス・ドライバの登録と密接な関係がありますので，登録に関するところのみ簡単に説明しておきます．

◆ イニシャル(初期化)処理

CONFIG.SYSファイルに登録されたデバイス・ドライバがMS-DOSに登録されるとき，**図17**(a)のように，本来このデバイス・ドライバが常駐する領域にデバイス・ドライバのファイルすべてがロードされます．そして，イニシャル処理ルーチンがコールされます．ところが，イニシャル処理ルーチンはこのとき一回のみしかコールされませんので，通常，イニシャル処理の後，切りはなされてしまいます〔**図17**(b)〕．

このときの様子を**図18**に示します．スターを夢見る

〈図18〉イニシャル処理は1回しか使わないのですててしまう



デバイス・ドライバ一家が，MS-DOSプロのオーディションを受けにいきます(MS-DOSによる登録)．ここで，スカウト氏(イニシャル処理ルーチン)は，いろいろと手を尽くして，処理ルーチン君をスターとしました(MS-DOSへ登録完了)．そして，デバイス・ドライバ一家の足を引っ張らないために(メモリ食い虫にならないように)マネージャであるデバイス・ヘッダ君と処理ルーチン君がMS-DOSプロに残り，イニシャル処理ルーチン君は一人去っていきます．とんだ猿芝居なのですが，通常はこのように処理されます．

この，切りはなしの理由はメモリの節約が目的です．MS-DOSは，基本的に640Kバイトしかプログラム・メモリがありませんので，日本語変換処理などでメモリを食われる日本での利用においては非常に有効です．

以上，とりとめもなく説明しましたが，この章では，デバイス・ドライバの存在意義，その構造，MS-DOSへの組み込まれ方を理解していただければ幸いです．

コンピュータに関する学習は，ただ本を読むだけではなかなか理解しにくいものです．各デバイス・ドライバの例を参考にして自分だけのデバイス・ドライバを作ってみてください．より一層理解が深まると思います．

なお，MS-DOS 3.1においても**PRINT.SYS**などが**IO.SYS**からはずされたバージョンなどがあり，現在市販中のMS-DOS 3.3(A)を含め，本章の内容といくらか異なった仕様のものが市販されています．

◆参考文献◆

(1) MS-DOS3.1プログラマーズリファレンス　マニュアル，日本電気．
(2) 標準MS-DOSハンドブック，アスキー出版局，アスキー．
(3) MS-DOSデバイス・ドライバ活用技法，アスキー出版局，アスキー．

Keyword

BIOS　Basic Input Output Systemの略で，ハードウェアを直接アクセスする基本ソフトウェアのこと．OSの階層構造においては，一番下位(ハードウェア寄り)に位置する．パソコンでは，キーボード，CRT，BIOS，Disk BIOS，プリンタBIOSなどがある．

資源管理　コンピュータ・システムがもつ各種の資源(メモリ，補助記憶装置，入出力装置，各種のプログラム，データ，etc.)を効率よく有効的に利用するための管理．MS-DOSにおけるファイル管理や，実行プログラム・ロード時のメモリのアロケーションがこれにあたる．

一般的には，ほかに，タスク実行時におけるCPUやI/Oの負荷，メモリのタスクやI/Oへのアロケーション，メイン・メモリの占有度，I/Oの故障率判定など様々な資源管理が行われている．システム全体を考えた場合，人も資源の一つなので管理の対象となる．

仮想記憶　主記憶装置がもつ実際の主記憶容量(実記憶容量)よりも大きな容量の仮想的な主記憶容量(仮想記憶容量)をシステムに与える方式である．主記憶装置と大容量の補助記憶装置が組みになって仮想記憶空間(論理記憶空間)を実現する．

主記憶装置より大きな容量のプログラムでも，プログラマは意識せずに作成することができる．またマルチジョブの実行時には，補助記憶装置との間でロールイン・ロールアウト処理が行われるが，そのときのメモリへのプログラム，データの再配置を容易にする．

身近なところではOS/2がこの方式をとっている．

第2章 1 I/Oポートにつながる RAM ディスクを作る
ブロック型デバイス・ドライバとRAMディスクの作り方[ハードウェアの製作]

複数のデータをまとまって扱えるブロック型の実例として，RAM ディスクを製作し，そのデバイス・ドライバを作ります．可動部のない RAM ディスクは，ソフト開発の強いみかたです．

藤井 敦

16 ビット・パーソナル・コンピュータが普及してその利用・応用分野が広がりを増すにつれて，プログラムの開発環境もかなり整備されてきました．普及初期のような Basic によるプログラミング一辺倒の時代から，OS は MS-DOS になり，プログラミング言語は Fortran, Pascal, C などのコンパイラを使用したものへと変わってきました．

これに伴って，メモリ容量の増大や外部記憶装置(フロッピ・ディスクやハード・ディスクなど)の使用も一般的になりました．また，初期の頃の 5 インチ・フロッピ・ディスク装置の動作の遅さにはうんざりしたものでしたが，フロッピ・ディスク装置の機構の改善や，ハード・ディスク装置の使用によって，動作速度も一段と向上しました．

しかし，せっかちなプログラマはコンパイル速度などの更なる向上を求めているようです(筆者自身もその一人なのですが)．そもそも，プログラム開発時のコンパイルやリンクに費やされる時間は，たんなる待ち時間ですので，この時間が短縮されるということは歓迎すべきことです．

コーディング後にすぐ実行できる Basic のような環境を羨ましく思うこともあります．

● RAM ディスクの必要性

ところで，第 1 章でも述べてあるように，MS-DOS にはユーザが作成したデバイス・ドライバを組み込むことが可能になっています．

これにより，コンピュータのメモリの一部やメモリを追加するなどして超高速な RAM ディスク装置を搭載することが可能となり，見かけ上のコンピュータの動作速度を向上することもできます．また，新しい外部記憶装置(光ディスク，カード・メモリなど)を接続して，専用のデバイス・ドライバを組み込むことにより，MS-DOS のコマンド・レベルでそれらの装置をアクセスしたり，ユーザ・プログラムの中でもフロッピ・ディスクの場合と同様にファイルとして扱うことが可能となります．

MS-DOS がこれまで普及し，応用されている理由の一つには，このデバイス・ドライバにあるような気もします．

さて，昨年あたりまでは，メモリ・デバイスの供給不足もあり，とくに大容量 DRAM の入手が困難でしたが，最近になって製品の入手も比較的楽になったようです．

そこで，ブロック型デバイス・ドライバの製作例として，1 M バイト DRAM を 16 個使用した 2 M バイト RAM ディスクについて紹介します．

RAM ディスクとは

ディスク装置といえば，フロッピ・ディスクやハード・ディスクのように回転磁気記録円盤を使用した書き込み/読み出しが可能な外部記憶装置を一般的に思いうかべますが，「RAM ディスク」とはその名のとおり磁気記録円盤の代わりに半導体メモリ(RAM)を使用したものです．

機構系の多いフロッピ・ディスクやハード・ディスクに比べ，時間のかかるヘッド・シーク・タイムやヘッド・セトリング・タイムがゼロであることと，データの転送速度がメモリとのアクセス・タイムのみによって決定されることなどから，非常に高速なディスク・アクセスが可能となります．

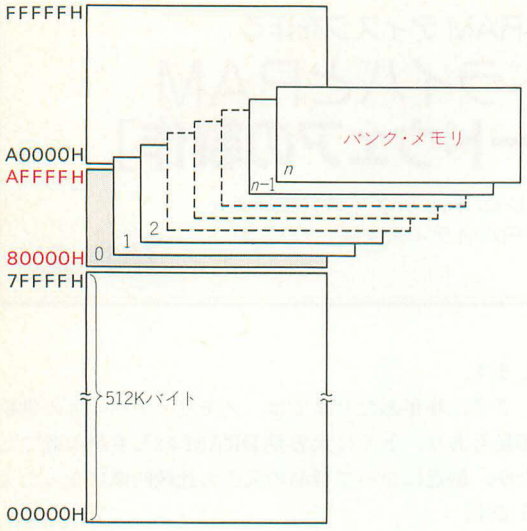
● RAM ディスクをどこに使うか

この高速性を活かす利用法としては，何度も繰り返し使用されるファイルの格納ディスクとして利用するときに効果を発揮します．例えば，コンパイラによるプログラム開発時には，ソース・ファイルやライブラリ・ファイル，そして一時ファイルの格納に使用することにより，驚くほどの高速でコンパイルを終了することができます．このことは，待ち時間が短いということで，いらいらもなくなりますし，何よりも精神衛生上の効果も見逃せません．

また，パーソナル・コンピュータをワード・プロセッサとして使用する場合の，辞書ファイルの格納ディスクとしての使用も考えられます．

しかし，このような便利な RAM ディスクにも欠点はあります．それは，電源を切るとその内容がすべて

〈図1〉PC9801の一般的なバンク・メモリの構成



失われてしまうことです．また，プログラムの暴走などによるファイルの破壊も考えられます．このため，少なくとも重要なファイルだけは，頻繁にフロッピ・ディスクにコピーしておくことを怠ってはいけないでしょう．

しかし，これらの欠点があるにせよ，「RAMディスク」の高速動作による効果には大変な魅力を感じます．

RAMディスクの仕様

まず，ここで製作する「RAMディスク」の主な仕様を紹介します．RAMディスクに限らず，何かを製作する場合に，その装置の仕様を決定することは大変重要なことです．

何に接続するのか，どういう使い方をするのかなどを熟考して決定しなければなりません．これによって，中途半端な物を作ってしまうことを防ぐことができます．

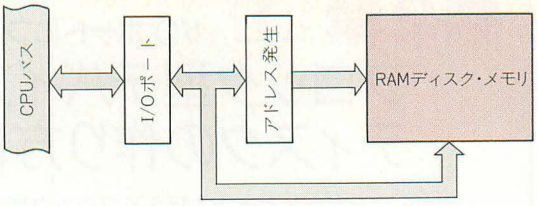
さて，RAMディスクの仕様決定には次のような事を考えなければなりません．

① 接続するパーソナル・コンピュータは？

RAMディスクは，それ単体では何の動作もしません．パーソナル・コンピュータと接続して初めて使用可能となります．そこで，どの機種のパーソナル・コンピュータに接続して使用するのかを決めなければなりません．理想的には，どの機種にでも接続できるというのが良いのかもしれませんが，メーカ間のハードウェアの違いや信号タイミングの違いなどがあり，なかなか面倒なものです．

そこで，筆者自身が使用しているパーソナル・コンピュータがPC9801シリーズであることや，ハードウェア解説書の入手が容易なこと，そして一般的であるという理由から，PC9801シリーズ専用とすることにしました．

〈図2〉I/Oアクセス方式RAMディスクの概念



もちろん，わずかな変更によりほかの機種での応用も可能でしょう．

② RAMディスクのディスク容量は？

RAMディスクの容量を考える場合，どのような目的に使用するかによってその容量は変わってきます．先にも記したように，RAMディスクの特徴は高速な動作にありますので，この特徴を生かすことが主な目的となります．筆者の使用目的は主にプログラム開発時のソース・ファイル，ライブラリ・ファイルや作業用の一時ファイルなどの格納用として使用しています．その他にも日本語ワード・プロセッサの辞書ファイル用としての使用も考えられますが，一般的にはプログラム開発用としての使用が多いようです．

このような目的では，フロッピ・ディスク1枚分程度の容量で十分な場合がほとんどですが，少し規模の大きなプログラムの場合はファイルの数も多くなりますので，多少の余裕がほしいところです．ここで製作するRAMディスクでは，余裕を見て2Mバイトの容量とします．

③ バンク方式とI/Oアクセス方式

「RAMディスク」のメモリは，必ずしもCPUの管理するメモリ領域に置く必要はありません．これは，このメモリ領域にプログラムなどをロードして実行するのではなく，たんにデータを格納する目的だけだからです．また，フロッピ・ディスクやハード・ディスクと同じように，ブロック単位(セクタ単位)でデータの読み出し/書き込みができる機能があればよいわけです．

一般的に，RAMディスクのメモリ・アクセス方式には，バンク・メモリ方式とI/Oアクセス方式の2種類があります．

バンク・メモリ方式は，CPUの管理するメモリ領域にRAMディスクを置く方式です．図1にPC9801の場合のバンク・メモリの構成例を示します．PC9801は，CPUの管理するメモリ領域が1Mバイトですが，ユーザが使用できるメモリ領域は640Kバイトです．このユーザ・メモリ領域から128Kバイトをバンク・メモリとして切り出し，複数枚の128Kバイト・メモリを切り替えて使用し，大容量のRAMディスクを構

成するものです．

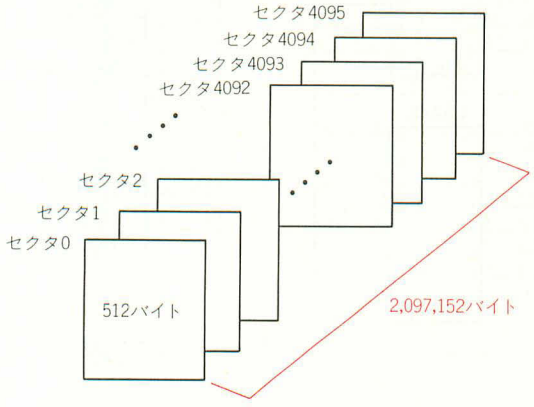
市販のバンク・メモリ方式のソフトウェアでは，この128Kバイトに0～255(最大)個のバンクの一つを割り当て，通常，メイン・メモリが640Kバイトになっています．いざRAMディスクが必要になれば，そこを別のバンクに切り替えて使います．そのため割り込みなどに制限があります．

これに対して，I/Oアクセス方式は文字どおりI/Oポートを通してディスク・メモリをアクセスする方式です．その概略を**図2**に示します．I/Oポートによってメモリを選択するためのアドレスを設定して，ディスク・メモリとのデータ入出力を行うものです．

この方式の欠点としては，ハードウェアが多少複雑になることと，バンク・メモリ方式に比べてアクセス速度が多少遅くなることです．

以上の主な仕様をもとに，RAMディスクを製作することにします．まず，デバイス・ドライバの作成の前に，説明をわかりやすくするためにハードウェアの説明から始めることとします．

〈図4〉RAMディスクの論理構成



RAMディスクのハードウェア

ここで製作するRAMディスクのハードウェア構成を，**図3**に示します．製作記事などで紹介される機会の多い増設メモリとは，多少異なった構成になっています．基本的なことは，CPUからRAMディスク中の任意の1バイトをアクセスすることがないということです．つまり，フロッピ・ディスクなどと同様に，ブロック単位(セクタ単位)で読み書きができればよいということです．このことにより，ハードウェアの構造も異なってきますし，簡略化を図ることも可能となります．

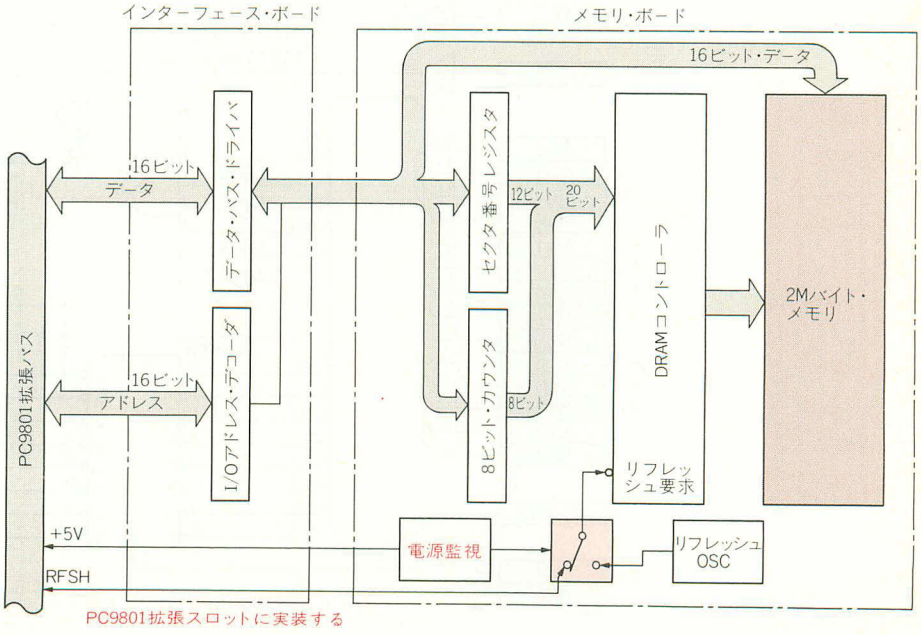
具体的には，**図4**に示すような論理構成とします．2Mバイト(2,097,152バイト)の容量を4096ブロックに分割して，1ブロックが512バイトの容量のセクタを構成します．このセクタには順に0～4095までの番号を付け，データの読み書きはセクタ単位で行います．

● 2枚のボードで構成する

さて，構成図で示したように，この「RAMディスク」は二つのボードによって構成されています．PC9801の拡張用スロットに実装するバス・インターフェース・ボードと，それに接続する2Mバイト・メモリ・ボードの2枚です．

最良の方法はすべてを1枚のボードに納めて，かつPC9801本体の拡張スロットに実装したかったのです

〈図3〉
RAMディスクの
構成概要



が，市販のユニバーサル・ボードではたくさんのICを実装できません．そのため，ラッピング配線を行うことによる実装スペースの増大や，修正のための予備スペースを確保できないため断念しました．

バス・インターフェース・ボードはデータ・バス・ドライバ部と，メモリ・ボードで使用するI/Oポート・アドレス・デコーダ部などによって構成されています．

また，メモリ部はメモリ・アドレス発生部，DRAMコントローラ，リフレッシュ制御部，それにメモリから構成されています．

DRAMコントローラは，数多くのLSIが発表されていますがここでは山下電気製のMCC1を，またDRAMは1Mビットの TC511000P10(東芝製)を使用しました．

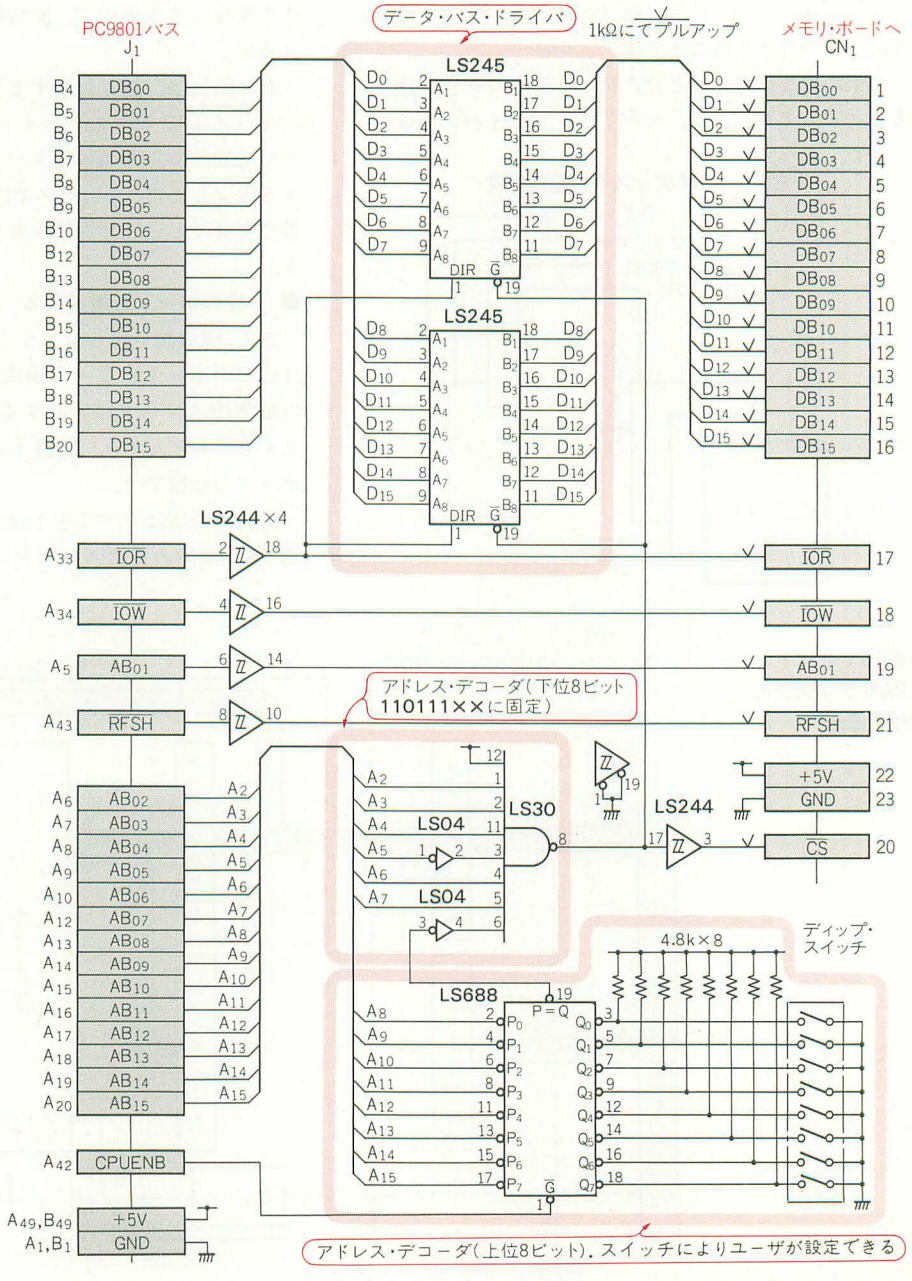
それでは，ハードウェアの主な部分について説明して行きます．回路図を図5および図6に示します．

TC511000　DRAMの概要

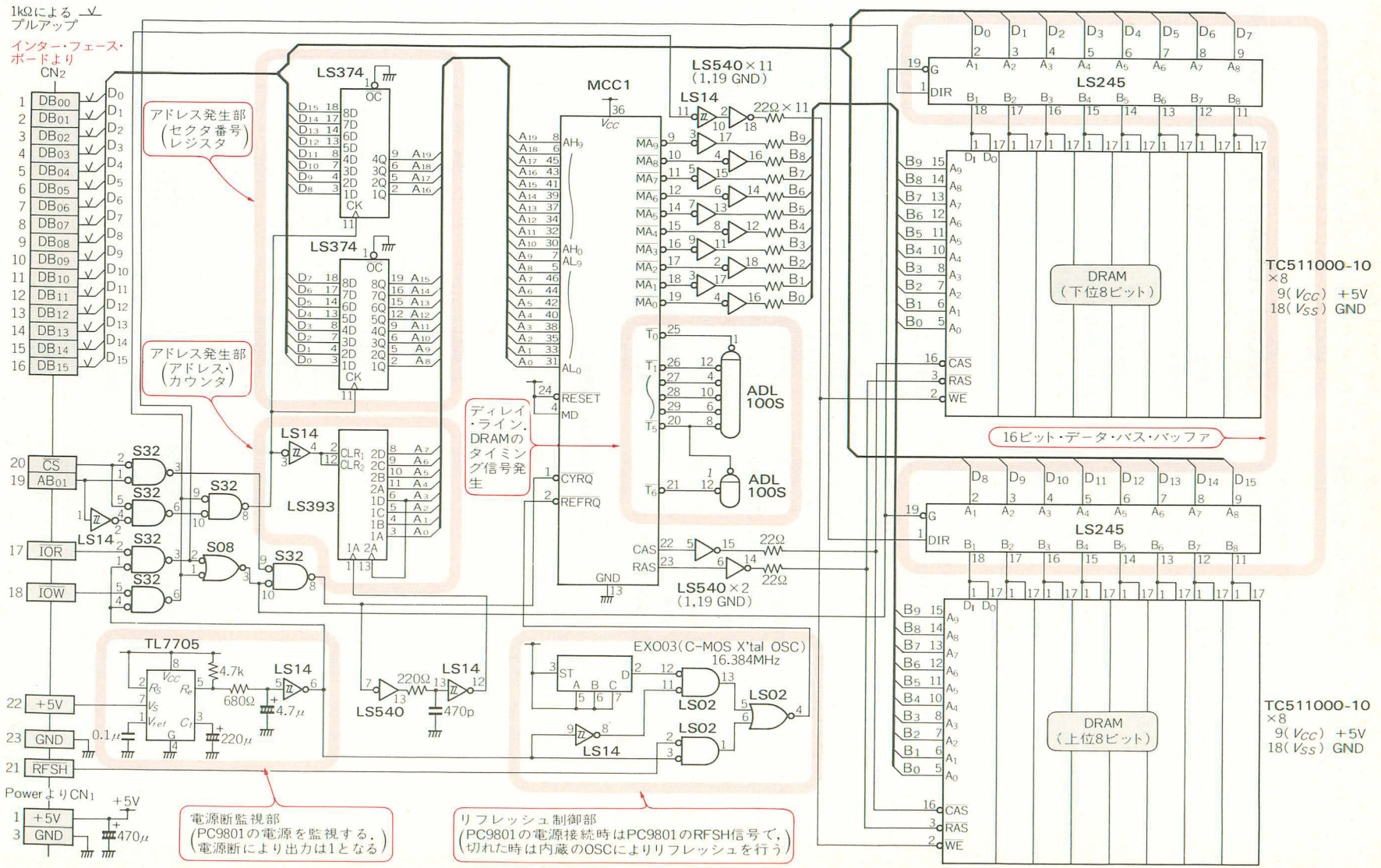
パーソナル・コンピュータのメモリの大容量化の中でつい最近まで256Kビット DRAMが主流であったのですが，2年程前に1Mビット DRAMが発表されて一気に1Mビット時代になってしまいました．

今回使用する1Mビット DRAMは，東芝から発売

〈図5〉
PC9801インターフェース・ボード



〈図6〉RAMディスクのメモリ・ボード



されているTC511000ファミリのもので，1,048,576×1ビット構成(512行×2048列)の単一5V電源で動作するものです．

アクセス・タイムは，100 nsタイプと120 nsタイプのものがありますが，本RAMディスクでは100 nsタイプのTC511000P10を使用しました．

● ヒドン・リフレッシュを使う

このDRAMは，ヒドン・リフレッシュが行えます．このリフレッシュは，リード・サイクルやアーリ・ライト・サイクル中に，まだ$\overline{CAS}$がアクティブ状態のままで$\overline{RAS}$を非アクティブにし，再び$\overline{RAS}$をアクティブにして$\overline{CAS}$ビフォア$\overline{RAS}$リフレッシュを行うものです．リフレッシュ・サイクルがリード・サイクルに隠れて行われるため，このような呼び名が付いているようです．

最近のDRAMでは，ほとんどの品種がヒドン・リフレッシュをサポートしており，またDRAMコントローラも同様にヒドン・リフレッシュ対応が可能となっています．また，$\overline{CAS}$ビフォア$\overline{RAS}$リフレッシュでは，内部に設けられているリフレッシュ・カウンタにより自動的にリフレッシュ・アドレスが発生されます．

これらの改良によりDRAM使用時のタイミング設計の面倒さから解放され，SRAM感覚でDRAMを使用することができるようになりました．

本RAMディスクでは，このTC511000P10を16個使用して2Mバイトのディスク容量としました．

TC511000P10のピン配置図と特徴を図7に示しますが，特性表やサイクル・タイミング図などは省略しますので，ほかの文献・製作記事や規格表など(参考文献の覧を参照 p.88)を参照してください．

MCC1　DRAMコントローラの概要

DRAMを使用する場合に，DRAMコントローラを使用することによって，回路設計や調整が大変楽になります．

DRAMコントローラは数多くのものが発表されており，現存するほとんどのCPUとインターフェースすることができます．

ここで用いたMCC1は，マルチプレクスした列アドレス，行アドレス，$\overline{RAS}$，$\overline{CAS}$信号およびメモリ・アクセスとリフレッシュの競合調停など，DRAMの使用に必要なすべての信号を，わずかな外付け回路で生成することができます．

また，ヒドン・リフレッシュをサポートしていますので，最新のDRAMのもっている機能を使用することができます．

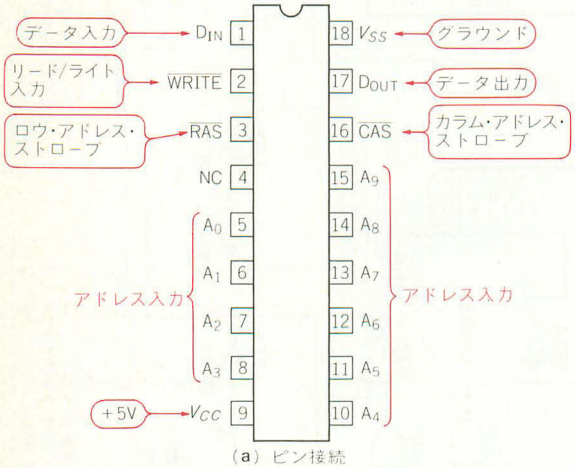
MCC1の端子接続図および特徴を図8に示します．

初期の頃の64 K, 256 K DRAMでは，リフレッシュのための内蔵リフレッシュ・カウンタをもっていないものもあるため，MD端子によって動作モードの選択を行うことができます．

● MD端子の設定

MD＝1に設定した場合は，DRAM内蔵のリフレッ

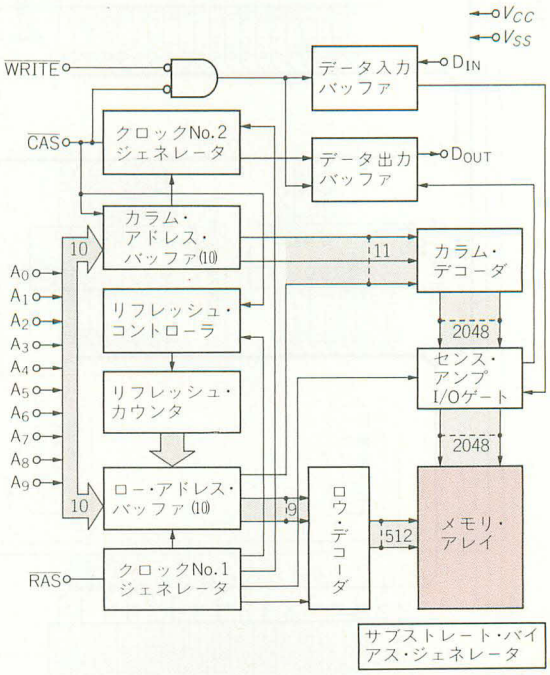
〈図7〉TC511000P/10のピン配置図とブロック図



(a) ピン接続

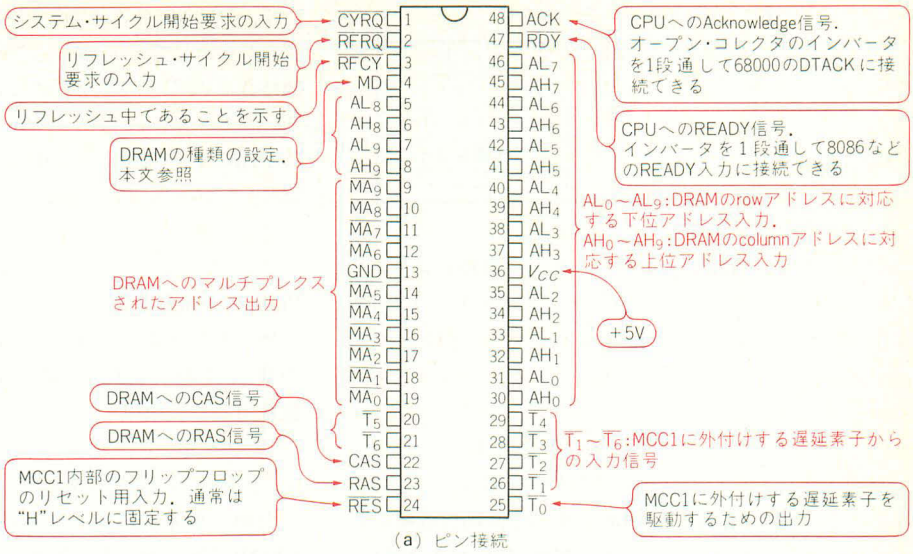
▶CMOS技術による低消費電力(動作時330mW)と高速動作が可能
▶アーリ・ライト(I/O共通動作)が可能
▶高速ページ・モードが可能
▶RASオンリ・リフレッシュ，CASビフォアRASリフッシュ，ヒドン・リフレッシュが可能
▶リフレッシュ方式は512/8ms
▶パッケージは18ピン・プラスチックDIP

(b) 主な特徴



(c) ブロック図

〈図 8〉
MCC 1 の端子接続図



(a) ピン接続

▶64K, 256K, 1Mビットの DRAMで使用可能
▶20ビットのアドレス入力と，10ビットのマルチプレクス・アドレスを出力する
▶外部アドレスによるRASオンリ・リフレッシュ，ヒドン・リフレッシュと，DRAM内蔵のリフレッシュ・カウンタによるCASビフォアRASリフレッシュとヒドン・リフレッシュをサポート
▶10ビットのリフレッシュ・アドレスを出力する
▶単一5V電源
▶形状は48ピンのプラスチックDIP

(b) 主な特徴

シュ・カウンタによって $\overline{CAS}$ ビフォア $\overline{RAS}$ リフレッシュ，およびヒドン・リフレッシュのみを行います．MD＝0 に設定した場合は，MCC1 の出力するリフレッシュ・アドレスによって $\overline{RAS}$ オンリ・リフレッシュおよび外部アドレスによるヒドン・リフレッシュを行います．

本 RAM ディスクで使用する 1 M ビット DRAM TC511000 は，リフレッシュ・カウンタを内蔵していますので，MD＝1 の動作モードで使用します．

また，$\overline{RAS}$/$\overline{CAS}$ などの信号は，外付けのディレイ・ラインによって生成されますので，使用する DRAM によって最適な遅延時間を選択する必要があります．本 RAM ディスクでは，20 ns タップのディレイ・ラインを使用しました．

RAM ディスク・ボード

図 5，**図 6** の回路図に沿って，RAM ディスク・ハードウェアの概要を説明します．

先にも説明したように，本 RAM ディスクは I/O アクセス方式を採用しました．また，RAM ディスクの場合，ブロック(セクタ)単位のアクセス(読み書き)ができればよいことになりますので，RAM ディスクの全容量である 2,097,152 バイト(2 M バイト)を 4096 セクタに分割し，1 セクタを 512 バイトとしてセクタ単位でアクセスします．

データの転送については，高速転送を確保するために 16 ビットのワード・データで行います．これにより，512 バイトのデータを 256 回の入出力で完了することができます．

◆ インターフェース・ボード

まず，インターフェース・ボードについて説明します．このボードは，PC9801 の拡張用スロットに挿入します．主な機能は 16 ビットのデータ・バス・バッファとアドレス・デコーダによって構成されています．アドレス・デコーダは，メモリ・ボードのハードウェア軽減と接続ケーブルの芯数を減らすために，インターフェース・ボード内に組み込みます．

PC9801 の場合，ユーザに解放されている I/O ポート・アドレスはたいへん少ないため，16 ビットをデコードし，**xxDCH, xxDEH**(16 ビットの I/O アクセスでは偶数アドレスのみ)を使用します．**xx** で示す上位 8 ビットは自由に設定することができますが，本 RAM ディスクでは **80H** に設定して **80DCH, 80DEH** でデバイス・ドライバ・プログラムを作成します．

I/O アドレス **80DCH** はデータの入出力ポートとして，**80DEH** はセクタ番号レジスタ・ポートとして使用します．**表 1** に I/O ポートのアドレスを示します．

◆ メモリ・ボード

メモリ・ボードは，インターフェース・ボードとケーブルによって接続して使用します．

本ボードはアドレス発生，DRAM コントローラ，2 M バイト・メモリおよびリフレッシュ制御の各ブロックによって構成されています．

〈表1〉I/Oポート・アドレス表

I/Oアドレス	$\overline{IOW}$	$\overline{IOR}$	機　能
xxDCH	0	1	データ・ライト
	1	0	データ・リード
xxDEH	0	1	セクタ番号ライト
	1	0	———

アドレス発生部は，12ビットのセクタ番号レジスタ(4096セクタ分)と8ビットのカウンタ(256ワード分)で構成されています．セクタ番号レジスタへのセクタ番号設定と同時に8ビット・カウンタをリセットし，カウンタはデータの入力，または出力のごとにIOR/IOWパルスによってインクリメントされます．

DRAMコントローラ部およびメモリ部については，回路的にも簡単ですし，一般的ですので詳細は省略します．必要があればほかの文献・製作記事(参考文献)などを参照してください．

◆ リフレッシュ制御部

最後にリフレッシュ制御部について説明します．RAMディスクは電源を切るとディスクの内容が破壊されてしまいますが，電源を投入したままであれば，停電などの不慮の電源断が発生するまではデータが保存されることになります．

● 電源の監視

本RAMディスクでは，電源をPC9801とは別電源とすることによりデータの保存を可能としました．従って，PC9801の電源断時のリフレッシュの問題を解決しなければなりません．このため，ハードウェアの簡略化を図る目的をも含めて，PC9801の電源が入っているときは，PC9801のリフレッシュ信号(実際は本体RAMがリフレッシュ中であることを示す$RFSH_0$信号)によって，電源が断のときはボード内の水晶振動子によってリフレッシュ・パルスを作成します．

これを実現するために，TL7705(TI製)によってPC9801の電源を監視して，リフレッシュ・パルスの切り替えを行います．TL7705はリセット・パルス発生用のICですが，端子の接続を変えることによって電源監視用として使用することもできます．

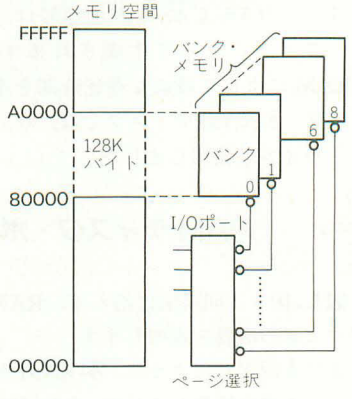
TC511000P10のリフレッシュ・サイクルは512/8 msですので，15.625 μsごとにリフレッシュ行えばよいため，水晶振動子［分周器を内蔵した水晶振動子EXO 03(キンセキ製)を使用］の16.384 MHzを256分周してリフレッシュ・パルスを作成します．

なお，PC9801から出力されるリフレッシュ信号($RFSH_0$信号)については，筆者が使用しているPC9801VXでは15.625 μsごとに出力されていますが，PC9801Fの場合は約28.3 μsごとに出力されているため，このままでは512/8 msのリフレッシュ・サイクルが確保されません．しかし，PC9801Fへの実装動作テストでは正常に動作しているようです．

以上で，ハードウェアについての説明を終わることにします．

Keyword

バンク・メモリ　パソコンPC9801ではCPUがリニアに管理できる最大のメモリ容量は基本的に1Mバイトです．しかし，扱うデータ量の増大やRAMディスクへ利用をするためにメモリ容量の増大化が図られており，1Mバイトのメモリ空間にさらにメモリを追加する手段としてバンク・メモリ方式が採用されています．図に示すように，メモリ空間の一部を切り出し，さらにそれをページ(バンク)分けしてメモリの追加を行います．メモリへのアクセスはI/Oポートによるページ選択で行いますので，プログラムなどを配置するリニアなメモリ空間としては使用できませんが，RAMディスクや画像フレーム・バッファなどへの応用には最適な方法といえます．



アクセス・タイムとサイクル・タイム　メモリ速度を示すために，アクセス・タイムとサイクル・タイムが使用されます．アクセス・タイムはメモリに対してアドレスなどの信号を与えてから読み出しデータが確立するまでの時間を表します．これに対してサイクル・タイムは，一つのアクセス・サイクルが終了して次のアクセス・サイクルに移れるまでの時間を表します．スタティックRAMでは，サイクルという区切りが存在しませんので，いつ信号が変化してもメモリの動作に支障はありませんが，DRAMでは，次のサイクルに入る前に“プリチャージ・タイム”が必要となります．一般的にDRAMでは，「サイクル・タイム≧アクセス・タイム＋プリチャージ・タイム」となります．

第2章 2 I/Oポートにつながる RAM ディスクを作る
ブロック型デバイス・ドライバとRAMディスクの作り方[デバイス・ドライバの構成]

フロッピ/ハード・ディスクなどはブロック型デバイス・ドライバが必要になります．
このデバイス・ドライバの詳細と RAM ディスク特有の問題点について検討します．

藤井 敦

第1章でデバイス・ドライバについて，基本的な説明がなされていますが，ここでは，実際のプログラミングに必要なデバイス・ドライバの構造について，具体的に説明していきます(本章では MS-DOS 2.1)．

MS-DOS の特徴として，MS-DOS 自身のシステム(**MSDOS.SYS** と **IO.SYS**)を修正することなく，ユーザが新たに作成したデバイス・ドライバを組み込むことによって，新たな周辺入出力装置が使用可能になるということがあります．たんにデバイス・ドライバを作成して，それを“**CONFIG.SYS**”ファイルに登録するだけでよいのです．

市販されている増設メモリ・ボードには，それを RAM ディスクとして使用するためのデバイス・ドライバが付属しているものも多くあります．また，市販の増設メモリの場合にはバンク・メモリ・アドレスやバンク選択ポート・アドレスなどが統一されており，デバイス・ドライバを共用して使用できるようにもなっているようです．

ここで紹介する RAM ディスクは，市販の増設メモリ・ボードとは異なって I/O アクセス方式ですので，デバイス・ドライバはユーザが作成しなければなりません．

デバイス・ドライバの位置づけ

MS-DOS システムは，**MSDOS.SYS** と **IO.SYS** によって構成されています．**MSDOS.SYS** は MS-DOS の中心であり，プロセス管理，メモリ管理，ファイル入出力など MS-DOS のほとんどの機能を処理します．また，アプリケーション・プログラムからのシステム・コール(**INT 21H**)を受けて，デバイスとの入出力を実行しますが，**MSDOS.SYS** の扱う範囲はデバイスという考え方までで，実際の入出力処理は行っていません．

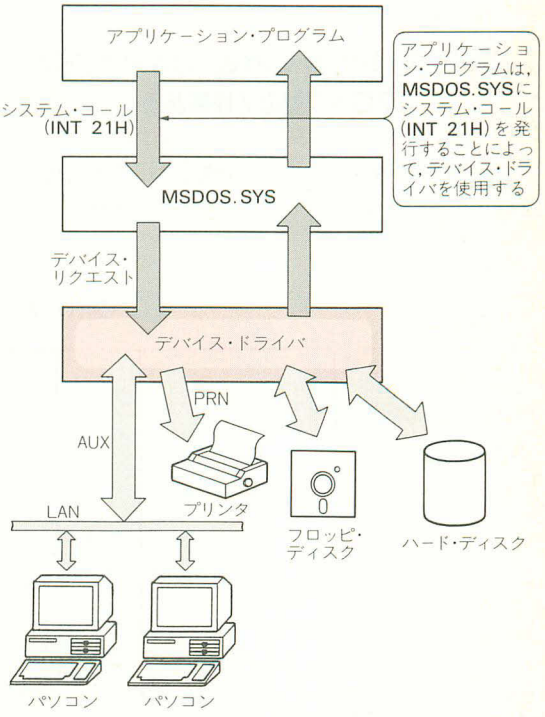
● IO.SYS の中に最低限必要なデバイス・ドライバが入っている

このデバイスを実現して実際の入出力を行うのがデバイス・ドライバなのです．MS-DOS ではバージョンによっては少し異なりますが標準で，コンソールとの入出力(CON)，プリンタへの出力(PRN)，RS-232C による補助入出力(AUX)，リアルタイム・クロック(CLOCK)，ディスクとの入出力の五つデバイスが使用可能になっており，これらのデバイス・ドライバは，まとめて **IO.SYS** に組み込まれています．つまり，**IO.SYS** は MS-DOS にあらかじめ組み込まれているデバイス・ドライバそのものなのです．

また，MS-DOS はユーザが新たに作成したデバイス・ドライバを組み込むこともできます．これにより，MS-DOS の標準のデバイス・ドライバ以外の入出力装置を使用可能としたり，既存のデバイス・ドライバに不満があれば，新しいデバイス・ドライバと置き換えることも可能です．これらユーザが新たに作成したドライバは，システムの立ち上げ時に組み込まれます．

図 9 に，MS-DOS におけるデバイス・ドライバの位

〈図 9〉デバイス・ドライバの位置づけ



置づけについて示します．

デバイス・ドライバの組み込み手順

MS-DOSが立ち上がるとき，ブート・プログラム(フロッピ・ディスクやハード・ディスクなどの特定のトラック・セクタに書き込まれている)によって，システム・プログラム(**IO.SYS**と**MSDOS.SYS**)がメモリ上にロードされます．まず，**IO.SYS**によってMS-DOSが標準で用意しているデバイス・ドライバの初期化を行い，使用可能な状態にした後で**MSDOS.SYS**に制御を移します．

ここで，**MSDOS.SYS**は“**CONFIG.SYS**”ファイルを読みに行きます．このファイルにユーザが作成したデバイス・ドライバが登録されていれば，そのデバイス・ドライバをメモリ上にロードして初期化を行い，使用可能とします．デバイス・ドライバは複数個あってもかまいません．

例えば，組み込みたいデバイス・ドライバのプログラム・ファイル名が“**RAMDISK.BIN**”とすると，“**CONFIG.SYS**”には，

DEVICE＝RAMDISK.BIN

の1行があればよいのです．また，**MSDOS.SYS**は“**CONFIG.SYS**”に複数の，

DEVICE＝……

DEVICE＝……

があれば，次々にそのデバイス・ドライバを組み込んで行きます．このときのメモリ・マップの状態を図10に示します．

デバイス・ドライバの構造にはキャラクタ/ブロック型の2種類ある

MS-DOSで扱うデバイスの種類には，その入出力の違いからキャラクタ型デバイスとブロック型デバイスの二つがあります．

キャラクタ型デバイスは，コンソールやプリンタ，あるいはRS-232C通信ポートなどのように一度に1バイトのデータを転送し，通常はランダム・アクセスのできないデバイスです．

これに対してブロック型デバイスは，フロッピ・ディスクやハード・ディスクのように，一度にまとまった量のデータをブロック単位で転送を行い，ランダム・アクセスが可能なデバイスです．

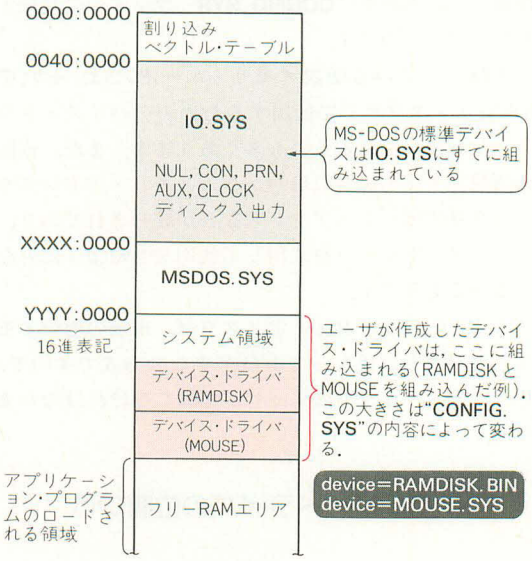
● RAMディスクはブロック型

従って，ここで紹介する「RAMディスク」は，ブロック型のデバイスに該当します．

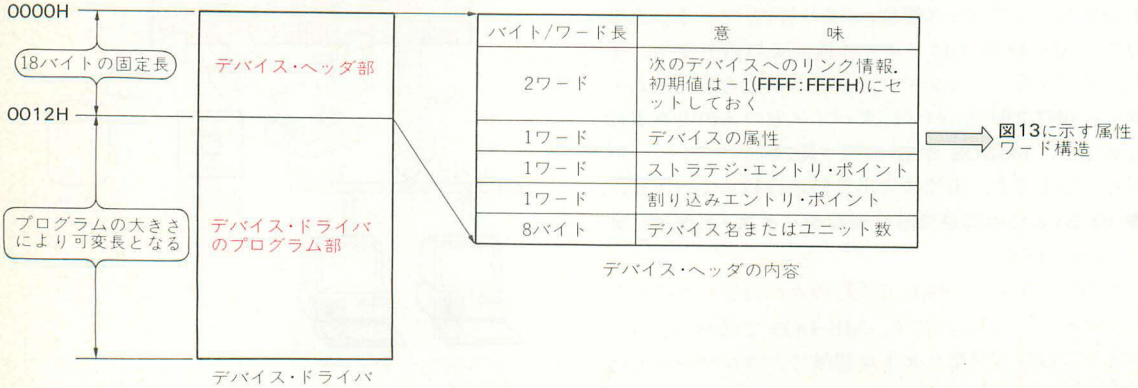
デバイス・ドライバの構造は，図11に示すようにデバイス・ヘッダ部とデバイス・ドライバ・プログラム部によって構成されています．デバイス・ヘッダ部はデバイス・ドライバの先頭に置かれており，**MSDOS.SYS**によるデバイス・ドライバの管理に使用されます．

デバイス・ドライバのプログラム部は，デバイスを

〈図10〉デバイス・ドライバ・ロード時のメモリ・マップ例



〈図11〉デバイス・ヘッダの構成



制御するためのプログラム本体で，デバイス・ヘッダの後に続いています．

デバイス・ヘッダの働き

デバイス・ヘッダは，MS-DOSがデバイス・ドライバを管理するための情報テーブルです．図11に示すように，この内容は，複数のデバイス・ドライバを連結するためのリンク情報や，キャラクタ/ブロック型の区別，デバイス・ドライバ・プログラムのスタート・アドレスなどの情報が格納されています．また，デバイス・ヘッダは，デバイス・ドライバのオフセット・アドレス**0000H**番地から作成されていなければならず，その領域は固定長で構造は決められています．

MSDOS.SYSは，アプリケーション・プログラムなどからのファンクション・コールを受け取った場合には，このデバイス・ヘッダによってデバイス・ドライバの識別やプログラム・アドレスを取り出して，デバイス・ドライバをコールします．

① リンク情報

デバイス・ドライバは，MS-DOSが標準で組み込むものとユーザが後から追加するものを合わせて複数個が組み込まれており，これらのデバイス・ドライバは一つのチェーンでリンクされて管理されています．

図12(a)に，MS-DOS標準のデバイス・ドライバのリンクの様子を示します．図に示すように，デバイス・ヘッダのリンク情報領域には，次のデバイス・ドライバのデバイス・ヘッダに対するポインタが格納されており，このポインタを見て行くことによって，すべてのデバイスを一つのリストとして扱うことができます．つまり，MS-DOSが必要なデバイスを検索する場合には，リンク情報をたよりにチェーンの先頭から順にたどって行くことによって，目的のデバイスを探しだすことが可能なのです．

また，ユーザが新たに組み込むデバイス・ドライバは，図12(b)に示すように標準のデバイス・ドライバの先頭(実際にはNULデバイスのすぐ後)に次々とリンクされて行きます．このときのリンク情報領域には，**MSDOS.SYS**によって新たなデバイス・ドライバを組み込むときに，その直後に組み込んだデバイス・ドライバの先頭アドレス(デバイス・ヘッダのアドレス)が書き込まれてチェーン接続されます．

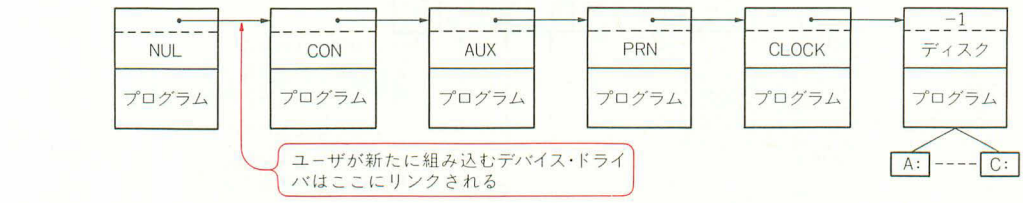
なお，ユーザが組み込むデバイス・ドライバのリンク情報領域の初期値には，－1を書き込んでおくことになっています．

● キャラクタ型の特徴

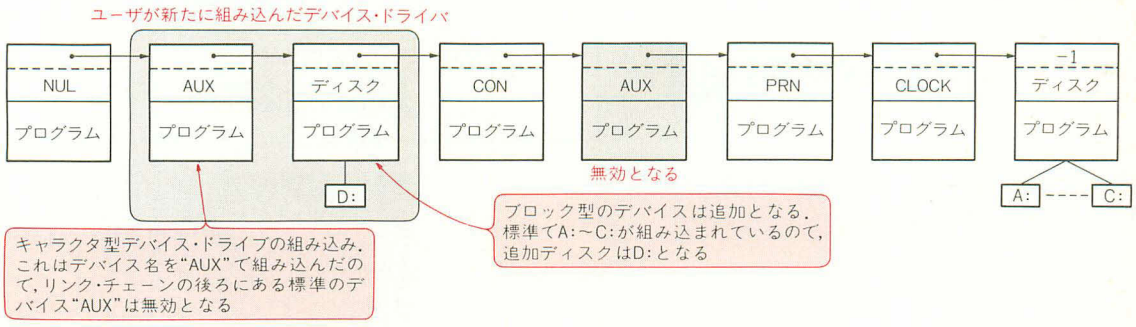
ところで，ユーザが組み込むデバイス・ドライバについて，キャラクタ型デバイス・ドライバの場合にはデバイス名によって各々が識別されているので，図12(b)に示すように，標準のデバイスと同じ名前のデバイスをユーザが組み込んだ場合には，後から組み込んだものが有効になり，元からあるもの(チェーンの後ろにあるもの)は無効となってしまいます．

従って，このことを利用して，“CON”，“AUX”などの標準のデバイス・ドライバが気に入らなかったり改良したい場合には，同一名のデバイス・ドライバを組み込んで標準のデバイス・ドライバと置き換えるこ

〈図12〉デバイス・ドライバのリンク構造



(a) MS-DOS標準のデバイス・ドライバのリンク



(b) ユーザによる新たなデバイス・ドライバの組み込み

とができます．代表的な例としては，日本語入力フロント・エンド・プロセッサがあります．これは，日本語入力のデバイス・ドライバを標準の“CON”デバイスと置き換えて実現しています．詳しいことは，第3章を参照してください．

● ブロック型の特徴

一方，ブロック型デバイス・ドライバではユーザが新たに作成したものを組み込む場合は，そのデバイスをシステムに追加することになります．つまり，標準のブロック型デバイス・ドライバと置き換えることはできません．

ブロック型のデバイスでは，ドライブ名を“A:”，“B:”，“C:”のように記号で参照しますので，例えば，すでに組み込まれている標準のブロック型デバイスが“A:”，“B:”，“C:”（ハード・ディスク，フロッピ・ディスク-1，フロッピ・ディスク-2など）の場合には，ユーザが新たに作成して組み込むドライブは“D:”，“E:”というように組み込み順に追加されて行きます．

② デバイスの属性

デバイスの属性は図13に示すように，このデバイスの型を示します．

③ ストラテジ・エントリ・ポイント

デバイス・ドライバ・プログラムのストラテジ・ルーチンのアドレスを格納します．ストラテジ・ルーチンは，**MSDOS.SYS**からのコマンド・パケットのアドレスを受け取るためのものです．

なお，この領域にはルーチンのオフセット・アドレスのみを格納します．セグメントは，デバイス・ヘッダと同一のセグメントであると見なします．

④ 割り込みエントリ・ポイント

デバイス・ドライバ・プログラムの割り込みルーチンのアドレスを格納します．割り込みルーチンは，コマンド・パケットによって指示されたコマンド処理を実際に実行するものです．

なお，この領域にはオフセット・アドレスのみを格納しますが，ストラテジ・エントリ・ポイントと同様にルーチンのセグメントは，デバイス・ヘッダと同一のセグメントであると見なされます．

⑤ デバイス名

キャラクタ型のデバイスの場合には，8文字までのASCII文字によるデバイス名が格納されます．

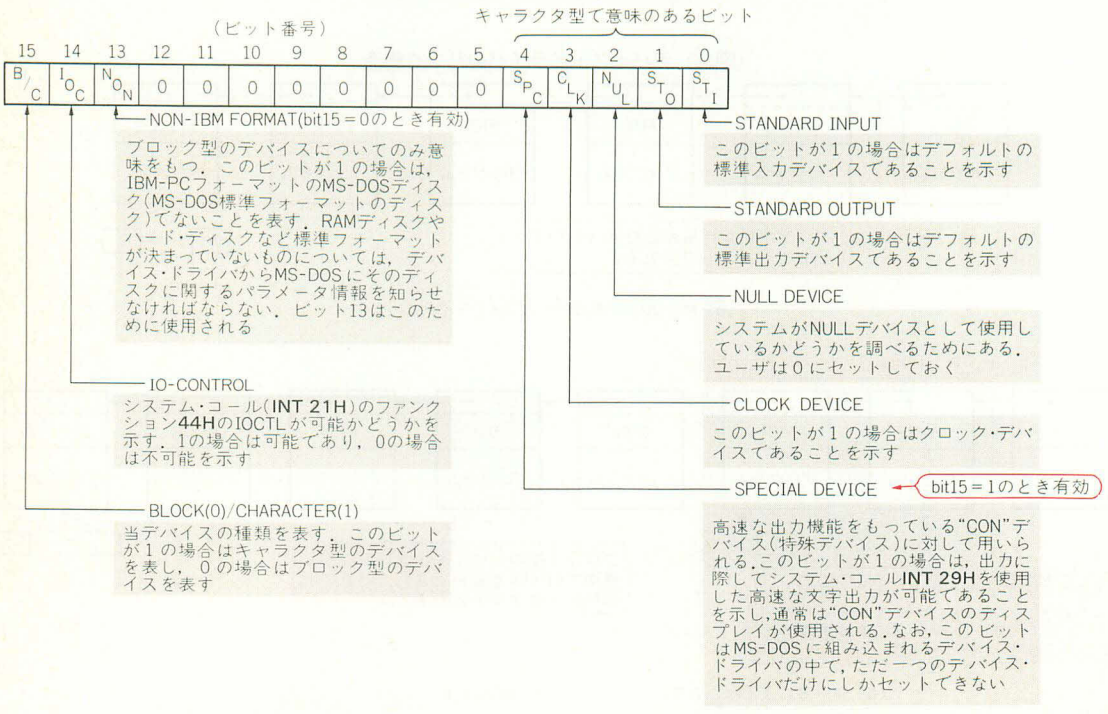
ブロック型のデバイスの場合は最初の1バイトのみが有効であり，このデバイス・ドライバで制御するデバイスの数を設定します．

デバイス・ドライバの呼び出し

図14に，**“MSDOS.SYS”**がデバイス・ドライバを呼び出して入出力を行う流れを示します．

“MSDOS.SYS”は，デバイス・ドライバに対して入

〈図13〉デバイス属性ワードの構造



出力を行う場合，まずコマンド・コードや各種のパラメータを設定したコマンド・パケット(後述)と呼ばれるコマンド・テーブルを作成します．そして，“MSDOS.SYS”は，このコマンド・パケットの先頭アドレスをレジスタES:BX(ES: セグメント，BX: オフセット)にセットして，まずデバイス・ドライバのストラテジ・ルーチンをFARコールします．

ストラテジ・ルーチンでは，このES:BX(コマンド・パケットのアドレス)の値をデバイス・ドライバ内の作業用のエリアにたんに退避するだけで，“MSDOS.SYS”にFARリターンします．

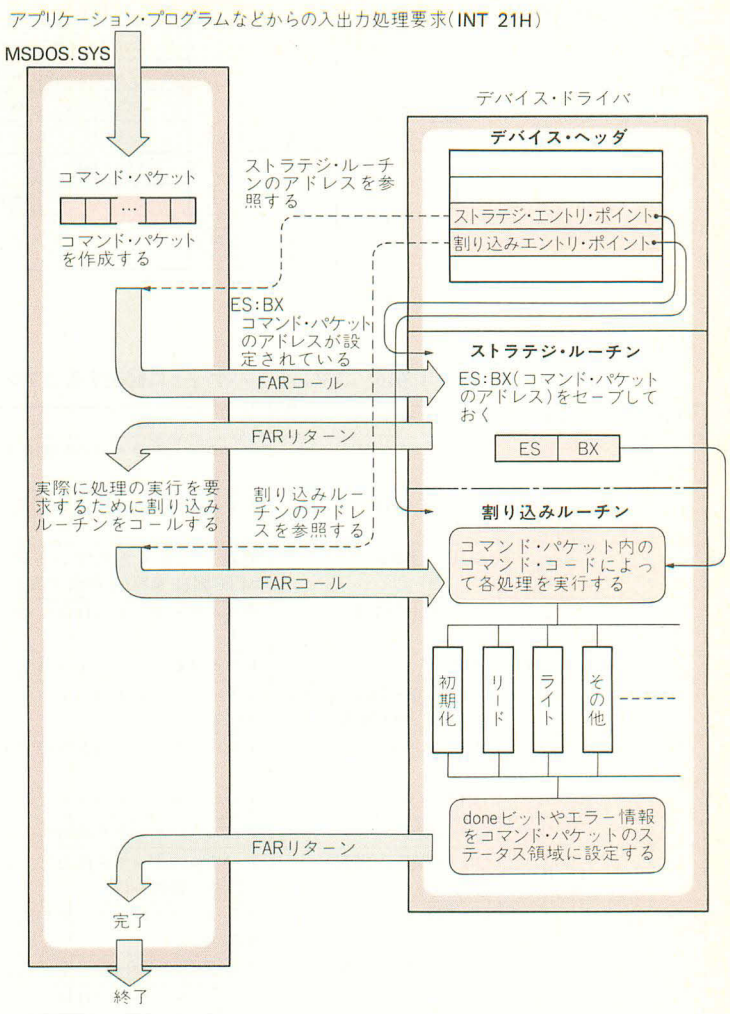
● 主要部分は割り込みルーチンに記述されている

この後，“MSDOS.SYS”はデバイス・ドライバの割り込みルーチンをFARコールして，実際の処理を要求します．割り込みルーチンでは，先に作業用エリアに格納したコマンド・パケットのアドレスを参照して，コマンド・コードや各パラメータを取り出してデータの書き込みや読み出しなど各々の処理を実行します．処理の終了後は，コマンド・パケット内のステータス・ワードに終了ステータス，エラー情報などをセットして“MSDOS.SYS”にFARリターンし，すべての処理を終了します．

デバイス・ドライバ側での処理においては，すべてのレジスタの値を保存しなければなりません．また，“MSDOS.SYS”が確保しているスタック領域は約20ワードですので，それ以上のスタックが必要な場合には，デバイス・ドライバ内に別途確保する必要があります．

さて，デバイス・ドライバの呼び出しにコマンド・パケットと二つのエントリ・ポイントを用いた，一見ややこしい方法を用いている意義については，「将来のマルチタスク版のMS-DOSを考慮しており，マルチタスクになった時でもI/Oリクエストの方法に関して大きな変更を加えないでもすむように」との説明が一般的に行われていました．しかし，マイクロソフト社が発表したマルチタスク・オペレーティング・システムOS/2のデバイス・ドライバでは，I/Oリクエストの考え方では共通なのですが，実際にはほとんど作り替えなければならないようです．

〈図14〉デバイス・ドライバのリクエスト・シーケンスの流れ



コマンド・パケット

前項でも説明したように，“MSDOS.SYS”がデバイス・ドライバを呼び出す場合，コマンド・パケットと呼ばれるテーブルに，コマンドやパラメータをセットしてデバイス・ドライバを呼び出します．

図15にコマンド・パケットの構造を示します．先頭の部分の13バイトは固定長でリクエスト・ヘッダと呼ばれ，すべてのコマンドに共通です．後の部分はパラメータ用の領域で可変長となっており，コマンドごとにその長さおよびその使われ方は異なります．

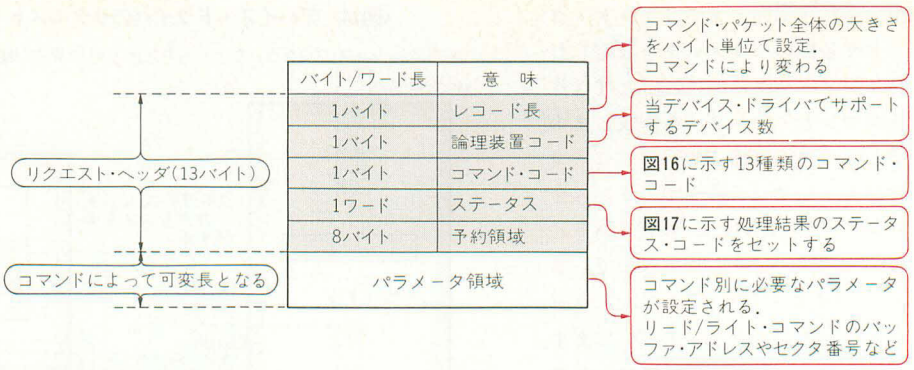
① レコード長

コマンド・パケットの総バイト長(リクエスト・ヘッダ長＋パラメータ領域長)が格納されています．

② 論理装置コード

ブロック型のデバイスに対してのみ意味があります．

〈図15〉
コマンド・パケットの
構造



〈図16〉コマンド・パケットに記述するコマンド・コード一覧

COMMAND	B/C	コマンド	機　　能	本RAMディスクでの処理
0	B/C	INIT	デバイス・ドライバの初期化を行う	FATとディレクトリ・エリアおよび予備セクタの初期化
1	B	MEDIA CHECK	ディスクが交換されたかを調べる	「メディア交換なし」のステータスのみを返す
2	B	BUILD BPB	現在アクセスされているディスクのBPB(図21参照)を作成する	BPBの先頭アドレスを返す
3	B/C	IOCTL INPUT	デバイス・ドライバ自身からデータを入力する	本デバイス・ドライバでは何もしない
4	B/C	INPUT	デバイスからデータを入力する	セクタからの読み出しを行う
5	C	NON-DESTRUCTIVE INPUT NO WAIT	入力バッファの先頭の1バイトの内容を調べる	キャラクタ型デバイス・ドライバでのみ有効(本デバイス・ドライバでは何もしない)
6	C	INPUT STATUS	入力バッファの内容が空かどうかを調べる	
7	C	INPUT FLUSH	入力バッファを空にする	
8	B/C	OUTPUT	デバイスへデータを出力する	セクタへの書き込みを行う
9	B/C	OUTPUT WITH VERIFY	デバイスへデータを出力し，正しく行われたかの検査も行う	
10	C	OUTPUT STATUS	出力バッファ内にデータが残っているかを調べる	キャラクタ型デバイス・ドライバでのみ有効(本デバイス・ドライバでは何もしない)
11	C	OUTPUT FLUSH	出力バッファの内容を空にする	
12	B/C	IOCTL OUTPUT	デバイス・ドライバ自身にデータを渡す	本デバイス・ドライバでは何もしない

＊B/C　B：ブロック型デバイス　　C：キャラクタ型デバイス

例えば，4台のデバイスをサポートしている場合には，0～3のいずれかの値となります．1台のデバイスしかサポートしていない場合は常に0となるので無視してもかまいません．

③ コマンド・コード

I/Oリクエストの種類を表すコマンド・コードが格納されます．I/Oリクエストの種類は13種ですので，0～12のいずれかの値となります．図16にコマンド・コード一覧表を示します．なお，I/Oリクエストの詳細については後で説明します．

④ ステータス

コマンドの処理結果を表すステータス・ワードを格納するための領域です．図17にステータス・ワードの構造を示します．コマンド処理が終わった後，doneビット(ビット8)に1をセットしてリターンします．

エラーが発生した場合は，errビット(ビット15)とdoneビットに1をセットし，さらに下位バイト(ビット0～7)にエラー情報をセットしてリターンします．

busyビット(ビット9)は，コマンド・コード6，10で使用するもので，キャラクタ型のデバイスでのみ意味があります．

⑤ 予約領域

将来，MS-DOSがマルチタスク版になったときのための予約領域とされています．

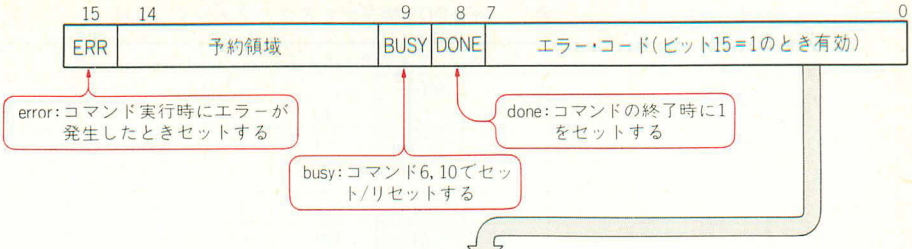
⑥ パラメータ領域

各コマンド特有のパラメータ情報が格納されます．詳しいことは，各コマンドの説明をするときに行います．

BPB(BIOS Parameter Block)

MS-DOSで扱うことのできるディスク装置は，フロ

〈図17〉
ステータス・ワードの構造およびエラー・コード



番号	エラーの内容
0	ライト・プロテクトされているディスクへの書き込み
1	存在しないユニット番号を指定した
2	ドライブの準備ができていない
3	コマンド番号の間違い
4	CRCエラー
5	コマンド・パケット長の間違い
6	シーク・エラー
7	メディア・タイプの間違い
8	セクタが見つからない
9	プリンタの用紙切れ
10	書き込みエラー
11	読み出しエラー
12	その他のエラー

〈図18〉BPB(BIOS Parameter Block)

バイト/ワード	意　味
1ワード	1セクタのバイト数
1バイト	1アロケーション単位当たりのセクタ数
1ワード	予備のセクタ数
1バイト	FATの数
1ワード	ルート・ディレクトリの数
1ワード	論理イメージのセクタ総数
1バイト	メディア・ディスクリプタ・バイト
1ワード	FATのセクタ数

▶MSDOS.SYSがディスクをアクセスするときのセクタ・サイズが何バイトであるかを設定する.
▶なお,ディスク装置の物理的なセクタ・サイズと一致している必要はないが,256バイトや512バイトのように32の倍数になっていなければならない

▶アロケーション単位とは,別名クラスタとも呼ばれるファイルへの割り当て単位を指す."MSDOS.SYS"ディスク上にファイルを作成する場合など,クラスタと呼ばれるものを単位として割り当てを行う.この1クラスタが何セクタで構成されるかを設定する.
▶なお,この値は2のべき乗(2^0～2^n)でなければならない.
▶一般的に,クラスタ・サイズを大きくすると小さなファイルをたくさん作るときにはむだが多くなり,小さくした場合はクラスタを管理するFATを大きくしなければならないが,その大きさにも限度がある.ちなみに,ごく一般的な5インチ2DDフロッピ・ディスクではセクタ・サイズが512バイト,クラスタ・サイズが1024バイトなので1アロケーション当たりのセクタ数は2(2^1)となっている

ディスクの先頭から何セクタをMS-DOSの管理から除外するかを設定する

▶FAT(File Allocation Table)は,ファイルがディスク上のどのクラスタを使用・連結しているかや空きセクタの位置などを記憶しているテーブル.このFATが何個用意されているかを設定する.
▶FATは非常に重要なため,一般的には二つ用意しておき,一つが読めなくなった場合の予備として使用する.しかし,RAMディスクの場合ではフロッピ・ディスクのように傷などによるセクタ不良がないので予備のFATを用意する意味はない.したがってFATの数は1

ルート・ディレクトリの総エントリ数を設定する

ディスクに含まれるセクタの総数(予備のセクタ数も含める)を設定する

本文に説明したメディア・ディスクリプタの値を設定する

▶一つのFATが何セクタを使用しているかを設定する.FATの各エントリは一つ当たり12ビット(1.5バイト)で構成されているので,この値はディスクの総容量(総クラスタ数)によって決定される.実際の設定方法については,本RAMディスクのBPB仕様設定を参照

ッピ・ディスクやハード・ディスクだけでなく,磁気バブル・メモリやRAMディスクのようなものまで,ブロック単位でランダムにアクセスすることができる記憶装置であれば,どのようなものでもかまいません.

また,このようなディスク装置のデバイス・ドライバを作成する場合にも,ディレクトリやFATの管理はすべて**MSDOS.SYS**が行うため,デバイス・ドライバではこれらについてまったく意識する必要はありません.

しかし,**MSDOS.SYS**が今どのようなタイプのディスク装置を取り扱おうとしているのかがわからなければ,ファイルのアクセスも不可能です.このために設けられたのがBPBと呼ばれるものです.

BPBは,ディスクに関する緒元を設定したメモリ領域で,デバイス・ドライバの中に設けられており,図18のような構造をしています.

● フロッピ・ディスクのフォーマット

ところで,フロッピ・ディスクの場合にはフォーマットの種類がたくさんあります.5インチのフロッピ・ディスクの場合でも,MS-DOSの標準フォーマッ

〈表2〉MS-DOS標準ディスク・フォーマット例

	3.5，5インチ・ディスク								8インチ・ディスク		
トラック数	80	80	80	(2DD) 80	40	40	40	40	77	77	(2HD) 77
セクタ/トラック	9	9	8	8	9	9	8	8	26	26	8
サイド数	1	2	1	2	1	2	1	2	1	1	2
セクタ・サイズ	512	512	512	512	512	512	512	512	128	128	1024
総容量(Kバイト)	360	720	320	640	180	360	160	320	250	250	1230
ディレクトリ数	112	112	112	112	64	112	64	112	68	68	192
セクタ/クラスタ	2	2	2	2	1	2	1	2	4	4	1
予約セクタ	1	1	1	1	1	1	1	1	1	4	1
FAT数	2	2	2	2	2	2	2	2	2	2	2
セクタ/FAT	2	3	1	2	2	2	1	1	6	6	2
FAT-ID	**0F8H**	**0F9H**	**0FAH**	**0FBH**	**0FCH**	**0FDH**	**0FEH**	**0FFH**	**0FEH**	**0FDH**	**0FEH**

トは10種類ほど存在します.

従って，デバイス・ドライバでは，一つのフロッピ・ディスクのためにBPBを何種類も用意しておかなければなりません．さらに，複数のBPBの中から，デバイス・ドライバが実際にアクセスしているフロッピ・ディスクの種類に合ったBPBを一つだけ選択するために，MS-DOSではフロッピ・ディスクのFAT(File Allocation Table)の先頭に書き込まれているFAT-IDによってその選択を行っています.

これにより，MS-DOSでは1台のドライブで何種類ものディスクを使用することが可能となっているのです．**表2**に，一般的なフロッピ・ディスクのMS-DOS標準ディスクにおけるFAT-IDを示します.

● **メディア・ディスクリプタ・バイトによって複数のBPBを判断する**

一方，**MSDOS.SYS**では，数あるBPBの中から現在使用されているのはどのBPBであるのかを簡単に知るために，BPBの中にメディア・ディスクリプタ・バイトと呼ばれる1バイトの値を設定しておきます．この値は，**00H**～**FFH**までの任意の値を設定することができますが，異なったBPBには異なった値を設定しなければなりません．値そのものに意味はなく，それぞれが異なった値をもっていることに意味があるわけです.

さて，RAMディスクの場合には一つのBPBしか存在しませんので，メディア・ディスクリプタ・バイトの値はとくに意味をもちません．従って，**00H**～**FFH**までの任意の値を設定してかまいません.

MSDOS.SYSではデバイス・ドライバから渡されたBPBをドライブの数だけ集めて，DPB(Drive Parameter Block)と呼ばれる作業領域を**MSDOS.SYS**の内部に作成し，**MSDOS.SYS**はこのDPBテーブルを参照してファイル・アクセスを行います.

I/Oリクエスト・コマンド

デバイス・ドライバで処理するコマンドは，**図16**に示したように13種類あります．各コマンドに対するコマンド・パケットの構造は，リクエスト・ヘッダについては固定長で，「コマンド・パケット」の項で説明したように各コマンド共通ですが，パラメータ領域についてはコマンドごとにそれぞれ決められています.

このパラメータ領域には，コマンド処理に必要なパラメータを**MSDOS.SYS**から渡されたり，コマンドの処理終了時には結果値をパラメータ領域にセットします．また，コマンドの終了ステータスをリクエスト・ヘッダのステータス領域にセットしなければなりません.

なお，ここではブロック型のデバイス・ドライバに関連するコマンドについてのみ説明することにします.

① INITコマンド （コマンド・コード＝0）

デバイス・ドライバの初期化を行うコマンドです．フロッピ・ディスクなどのデバイス・ドライバでは，デバイス・ドライバ自身の初期化やハードウェアの初期化が行われます．RAMディスクのデバイス・ドライバでは，このほかにFATエリアやルート・ディレクトリ・エリアの初期化が行われます.

このコマンドの終了時には，コマンド・パケットのパラメータ領域に，以下に示すような値をセットしなければなりません.

● **ユニット数**

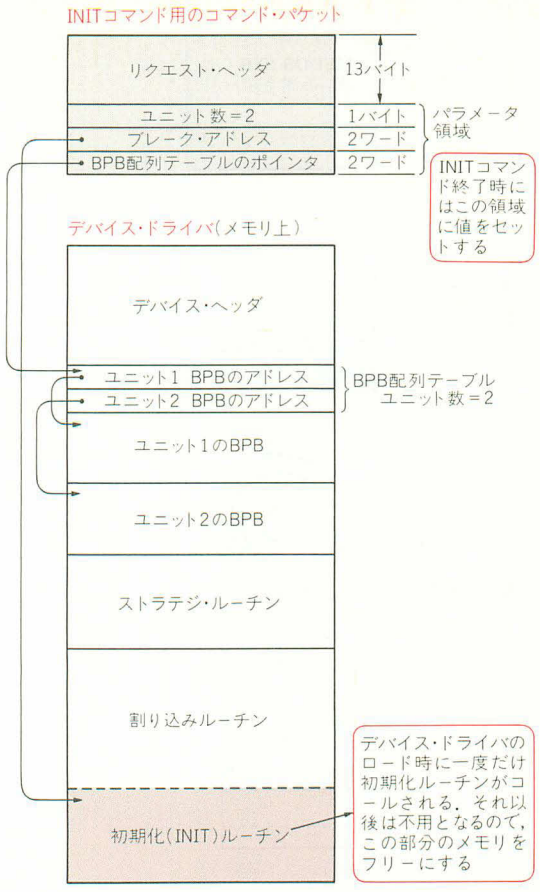
このデバイス・ドライバで，何台のドライブをサポートしているかを表す値を設定します.

● **ブレーク・アドレス**

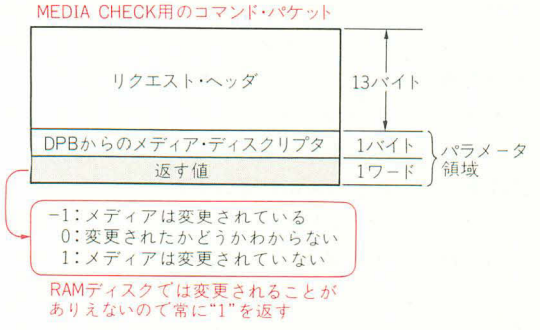
デバイス・ドライバ・プログラムが，どこで終わっているかを示すアドレスを設定します．INITコマンドは，デバイス・ドライバがロードされたとき一度しか実行されませんので，初期化の終了後には初期化ルーチンは不用となります.

従って，このルーチンをプログラムの最後部に置いておき，初期化終了時にはこのアドレスをブレーク・アドレスにセットすることによって，メモリを節約す

〈図19〉INIT コマンドの処理終了コマンド・パケットのセット



〈図20〉MEDIA CHECK コマンドのコマンド・パケット



ることもできます．

● BPB の配列へのポインタ

BPB へのポインタをユニット数だけ集めた配列テーブルのポインタを設定します．

以上の様子を図19 に示します．この図では，デバイス・ドライバでサポートするユニット数が 2 の場合について示しています．

② MEDIA CHECK コマンド
（コマンド・コード＝1）

ディスクが交換されたかどうかを調べるコマンドで，ブロック型のデバイスにのみ使用されます．本コマンドのコマンド・パケットを図 20 に示します．

コマンド・パケット・パラメータ領域の「DPB からのメディア・ディスクリプタ」には，調べようとするディスクのメディア・ディスクリプタ(**MSDOS.SYS** の DPB にセットされている BPB のメディア・ディスクリプタ)が設定されていますが，使い道はありません．

● 返す値

メディアが交換されたかどうかのチェック結果を，次の値でセットします．本コマンドでは，この値のみを処理結果としてセットします．

−1：交換された
0：交換されたかどうかわからない
1：交換されていない

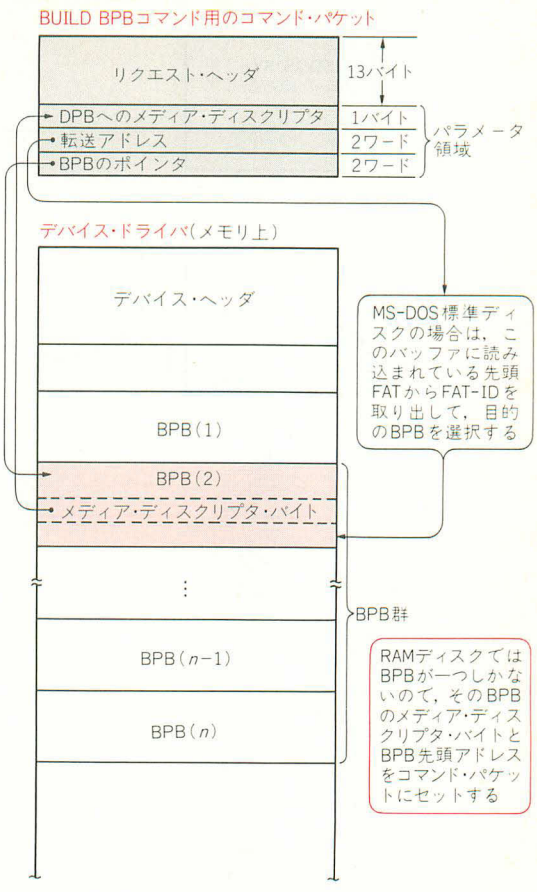
なお，RAM ディスクではメディアの交換はありえないので，常に“1”をセットします．

③ BUILD BPB コマンド
（コマンド・コード＝2）

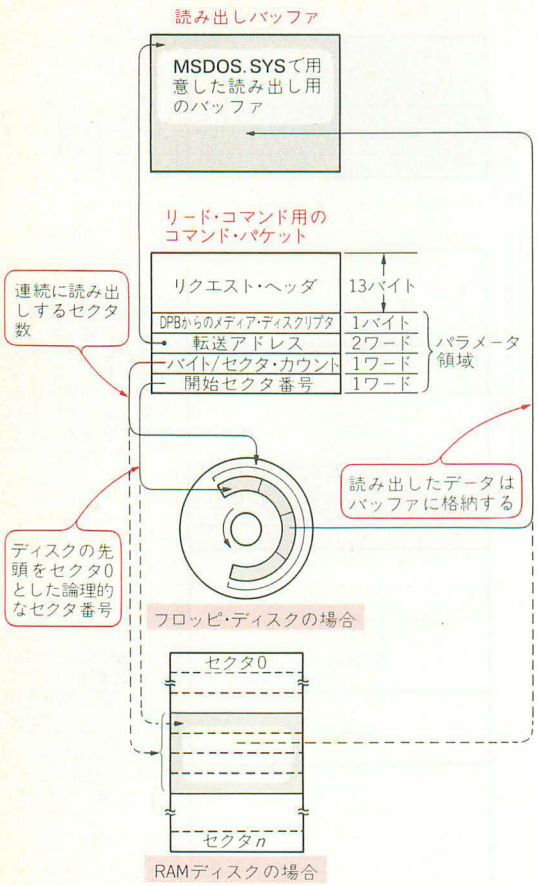
現在アクセスされているディスクの BPB を作成するコマンドです．

本コマンドのコマンド・パケットを図21 に示しま

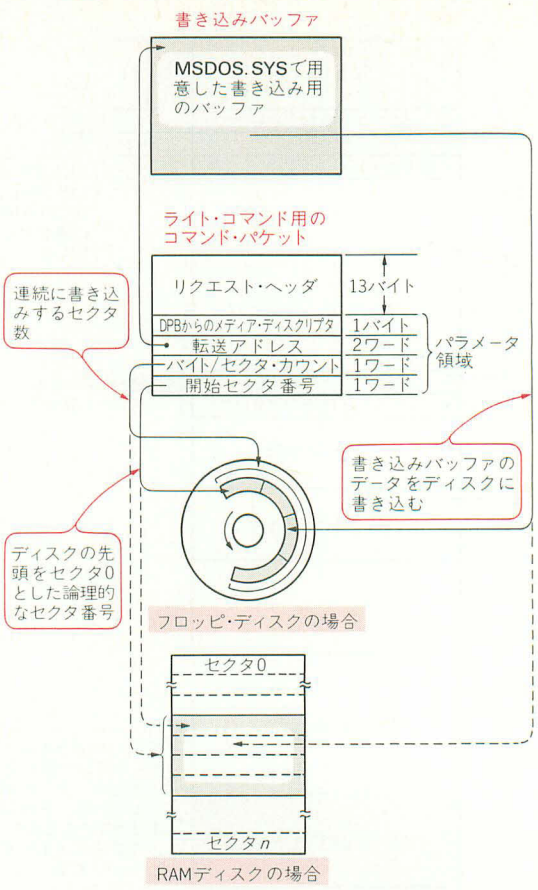
〈図21〉BUILD BPB コマンドのコマンド・パケット



〈図22〉リード・コマンドのコマンド・パケット



〈図23〉ライト・コマンドのコマンド・パケット



す．コマンド・パケットのパラメータ領域には，以下の値をセットします．

● DPBへのメディア・ディスクリプタ

作成あるいは選択されたBPBのメディア・ディスクリプタ・バイトを設定します．

● BPBに対するポインタ

作成あるいは選択されたBPBの先頭アドレスをセットします．

なお，MS-DOSの標準ディスク(デバイス・ヘッダのアトリビュート・ワードのビット13が0のデバイス)しか扱わないデバイス・ドライバに対しては，**MSDOS.SYS**がフロッピ・ディスクのFATの第1セクタを読み出して，そのバッファの先頭アドレスを「転送アドレス」にセットしてきます．これは，FATの第1バイト目に書かれているFAT-IDによって標準のフロッピ・ディスクのBPBを選択することが可能だからです．

RAMディスクの場合はBPBが一つしかありませんので，このBPBのメディア・ディスクリプタと先頭アドレスをパラメータ領域にセットします．

④ リード・コマンド
(コマンド・コード＝3，4)

このコマンドは，ディスクからのデータ読み出しと，デバイス・ドライバ自身からのIOCTLの読み出しを行うものです．二つのコマンドともコマンド・パケットが共通ですので，まとめて説明します．

本コマンドのコマンド・パケットを図22に示します．なお，この図ではコマンド・コード＝3のデータ読み出しについてのみ示しています．

コマンド・パケットのパラメータ領域には，コマンドの処理に必要な値がセットされています．コマンド・パケット・パラメータ領域の「DPBからのメディア・ディスクリプタ」は，MEDIA CHECKコマンドで説明したものと同じですが，あまり使い道はありません．

● 転送アドレス

データを格納するバッファの先頭アドレスがセットされています．ディスクから読み出したデータやIOCTL文字列は，このバッファに格納します．

● バイト/セクタ・カウント

読み出すIOCTL文字列の長さ(コマンド＝3)，または，読み出すセクタ数(コマンド＝4)がセットされています．

また，コマンドの処理終了時には実際に読み出した

バイト数，または，セクタ数をこの領域にセットしなければなりません．なお，ディスクからの読み出しでは，読み出しエラーが発生しない限りセクタ・カウント分のデータは読み出されており，コマンド要求時のカウント値がそのまま残っているので，改めて値をセットする必要ありません．

● 開始セクタ番号

読み出しを開始する，論理的なセクタ番号がセットされています．これは，ディスクの先頭セクタを番号0として順番に付けられた番号です．

⑤ ライト・コマンド
（コマンド・コード＝8，9，12）

これらのコマンドは，ディスクへのデータ書き込みと，デバイス・ドライバへのIOCTLの書き込みを行うものです．二つのコマンドともコマンド・パケットが共通ですのでまとめて説明します．

本コマンドのコマンド・パケットを**図23**に示します．なお，図ではコマンド・コード8のデータ読み出しについてのみ示しています．コマンド・コード9については，書き込んだ後のベリファイが付加されたものです．

コマンド・パケットのパラメータ領域には，コマンドの処理に必要な値がセットされています．コマンド・パケット・パラメータ領域の「DPBからのメディア・ディスクリプタ」は，MEDIA CHECKコマンドで説明したものと同じですが，あまり使い道はありません．

● 転送アドレス

書き込みデータが格納されているバッファの先頭アドレスがセットされています．ディスクへの書き込みデータやIOCTL文字列は，このバッファから取り出します．

● バイト/セクタ・カウント

書き込むIOCTL文字列の長さ(コマンド＝12)または，書き込むセクタ数(コマンド＝8，9)がセットされています．

また，コマンドの処理終了時には，実際に書き込んだバイト数または，セクタ数をこの領域にセットしなければなりません．なお，ディスクへの書き込みでは，書き込みエラーが発生しない限りセクタ・カウント分のデータ書き込みは行われており，コマンド要求時のカウント値がそのまま残っているので，改めて値をセットする必要はありません．

● 開始セクタ番号

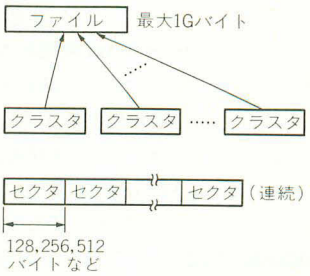
書き込みを開始する論理的なセクタ番号がセットされています．これは，ディスクの先頭セクタを番号0として順に付けられた番号です．

Keyword

システム・コール　ユーザ・プログラムの作成では，「キーボードからの入力」，「CRTへの出力」，「ファイルの入出力」などが頻繁に使用されますが，これらはハードウェアと密接に関連しており，一つ一つ記述するのは大変です．MS-DOSに限らずOSにはこれらの処理を実行するファンクション・プログラムがすでに組み込まれています．MS-DOSでは**MSDOS.SYS**内に72種類(Ver 2.0)ものファンクションが組み込まれており，ユーザ・プログラムから自由にコールして使用することができます．これをシステム・コール(ファンクション・コール)と呼んでいます．C言語の標準関数なども，このシステム・コールを使用して記述されています．

ブート・プログラム　パソコンでは，MS-DOSなどのシステム立ち上げをディスクから行います．このためにはディスクに格納されているシステム・プログラム(OS)をメモリ上にロードして実行しなければなりません．このロードと実行を行うプログラムを，ブート・プラグラム(正式にはブートストラップ・ローダ)と呼んでいます．ブート・プログラムはROM化されており，PC9801では，電源ONの後一定時間内(数秒間)にディスクがなければROM版BASICを，ディスクがある場合は特定のセクタからシステム・プログラムをロードして実行します．

クラスタ　MS-DOSでは，ファイルをクラスタという単位で管理しています．フロッピ・ディスクやハード・ディスクなどという装置の違い，あるいはセクタという物理的単位を意識せずにより統一的にディスクなファイルを扱うための単位です．図に示すようにクラスタは連続したセクタによって構成されていますが，ファイルを構成するクラスタは連続しているわけではありません．そのために，ファイル割り付けテーブル(FAT：File Allocation Table)によって，クラスタがどのような順につながってファイルを構成しているかを管理しています．



第2章 ③ I/Oポートにつながる RAM ディスクを作る
ブロック型デバイス・ドライバとRAMディスクの作り方[プログラムの作成]

RAM ディスクは，実際のフロッピ・ディスクとは異なり，データ部の初期化などは独自に行わなければなりません．プログラム・リストには詳しいコメントを入れて解説しました．

藤井 敦

デバイス・ドライバについての説明が長くなってしまいました．本題のデバイス・ドライバ・プログラムの作成について説明していきます．

最近のプログラム開発は，C 言語など高級言語を用いて行うことが一般的になりましたが，デバイス・ドライバについてはアセンブラによる作成が一般的なようです．日本語入力フロント・エンド・プロセッサ(日本語入力用のデバイス・ドライバ)など大規模なデバイス・ドライバでは，C 言語が用いられている例もあります．

しかし，セグメント配置が一般のアプリケーションと異なるなどの問題もあり，手軽に使用できるわけでもありません．また，プログラム・サイズが大きくなってしまいます．

ここで紹介する RAM ディスクような小規模のデバイス・ドライバでは，アセンブラで作成するほうが効率的です．

RAM ディスクの仕様設定

ハードウェア仕様の項でも説明しましたが，本 RAM ディスクでは総容量の 2 M(2,097,152)バイトを，1 セクタ当たり 512 バイトの物理的セクタ・サイズで分割して，セクタ総数を 4096 セクタで構成しています．

RAM ディスクのデバイス・ドライバは，セクタ単位でデータの読み書きを処理しますので，セクタ・サイズが BPB のパラメータ設定に大きく関わってきます．

表 3 に本 RAM ディスクの仕様を示しますが，表に従って BPB の各パラメータ設定について説明します．

① 1 セクタのバイト数(論理的なセクタ・サイズ)

物理的なセクタ・サイズと同一の 512 バイトとしました．ディスクの総容量が数十 M バイトと大容量の場合は大きくしなければなりませんが，本 RAM ディスク程度では 512 バイトが適当でしょう．

② 1 アロケーション当たりのセクタ数

MSDOS.SYS がファイルを扱う時のアクセス単位ですが，先に説明したようにメモリのむだを減らすために 512 バイトとしました．従って，セクタ数は 1 とします．

③ 予備のセクタ数

RAM ディスクの場合，システム・ディスクとして使用することはありませんので基本的には 0 でよいのですが，データの保存機能とイニシャル処理の機能拡張のために 1 セクタを使用します．これは，ハードウェアの項で説明したように，本 RAM ディスクではメモリ内容の保存のために電源を PC9801 本体から供給されるのではなく別電源としました．

これにより，PC9801 の電源を切っても，RAM ディスク内のデータは保存されています．しかし，メモリ内容の保存は行われていても，MS-DOS 立ち上げ時にデバイス・ドライバがロードされるごとに RAM ディスクの初期化プログラムにより FAT やディレクトリがクリアされたのでは，データを保存している意味がありません．

そこで，RAM ディスクの先頭セクタ(予備セクタ：セクタ 0)に特定のチェック・ワードを書き込んでおき，デバイス・ドライバの初期化ルーチン実行時にそのチェック・ワードが保存されているかどうかをチェックします．これにより，保存されている場合は RAM ディスクの初期化処理を行わず，破壊されている場合にのみ初期化処理を実行します．予備のセクタは，この目的のためのデータ保存チェック用セクタとして使用します．

④ FAT の数

FAT の数は 1 に設定します．これは，フロッピ・デ

〈表 3〉本 RAM ディスクの主な仕様

1 セクタのバイト数	512
1 アロケーション当たりのセクタ数	1
予備のセクタ数	1
FAT の数	1
ルート・ディレクトリ数	128
総セクタ数	4096
メディア・ディスクリプタ・バイト	00H
FAT に必要なセクタ数	12

ィスクなどとは異なり，RAMディスクの場合には一部のセクタのみが不良になってアクセスできなくなるようなことがないために，予備のFATを用意しておくメリットがないためです．

⑤ ディレクトリ数

ルート・ディレクトリの総数ですので，あまり多くする必要はありません(必要ならばサブ・ディレクトリを作成すればよい)ので128とします．

⑤ 総セクタ数

総セクタ数は，物理的に4096セクタと決定されます．

⑦ メディア・ディスクリプタ

本デバイス・ドライバで扱うデバイス(ユニット)の数は一つですので，**00H～FFH**までの値であれば自由に設定することが可能ですが，ここでは**00H**とします．

⑧ FATに必要なセクタ数

本RAMディスクではクラスタ・サイズ(1アロケーションのサイズ)＝セクタ・サイズですから総クラスタ数は4096となり，これを管理するためのFATのセクタ数は12となります．

これは，**MSDOS.SYS**がファイルの管理を行うために，クラスタに番号を付けて管理していますが，FATでは一つのクラスタの番号を12ビットで表しているため，4096のクラスタを管理するためには，

4096×12(ビット)＝49,152(ビット)

が必要となります．従って，FATに必要なセクタ数は，

49,152(ビット)÷8(ビット)÷512(バイト)
＝12(セクタ)

となります．

ところで，12ビットで表すことのできる最大値は4096ですので，**MSDOS.SYS**で管理することのできるクラスタ数は最大で4096ということになります．従って，RAMディスクの容量を2Mバイトより大きくする場合には，セクタ・サイズあるいは1クラスタ当たりのセクタ数(1アロケーション当たりのセクタ数)を大きく設定して，総クラスタ数を4096以内に収める必要があります．

プログラムの構造

本RAMディスクのデバイス・ドライバ・プログラムをアセンブルした「**RAMDISK.LST**」を**リスト1**(p.89)に示します．300行程度のプログラムですので，さほど難しいものではありません．以下に各ルーチンについて説明します．なお，BPBについては仕様の項で説明したとおりです．

① デバイス・ヘッダ

デバイス・ヘッダは，プログラムの先頭(**0000H**)で定義しなければなりません．定義内容は，「デバイス・ヘッダの構造」で説明したとおりですので，ここでは「デバイスの属性」ワードについて説明します．

これまでに説明したように，RAMディスクはブロック型のデバイスであり，MS-DOSの標準ディスクではありませんのでNON-IBM FORMATとなります．

また，IO-CONTROLについては，キャラクタ型のデバイスでは使用されることもありますが，ブロック型のデバイスではほとんど使用されませんので省略しました．

従って，「デバイスの属性」ワードは，

2000H(**0010000000000000B**)

に設定します．

② BPBの定義

表3のデバイス・ドライバの仕様のところで説明したとおりの内容です．

③ ストラテジ・ルーチン(strategy)

このルーチンは，レジスタES:BXにセットされてくるコマンド・パケットのアドレスを，作業用の領域(packet)に格納してFARリターンするだけです．

実際のコマンド処理は，割り込みルーチンのコールで実行されますが，そのときに必要なコマンド番号，パラメータはコマンド・パケットから取り出します．

④ 割り込みルーチン(interrupt)

このルーチンは，実際にコマンド処理を実行する入口となります．まず，デバイス・ドライバで使用しているすべてのレジスタを退避した後で，コマンド・パケットからコマンド・コードを取り出し，コマンド・ジャンプ・テーブル(**jmptable**)の内容にしたがって，該当する各ルーチンへ分岐します．

コマンドの処理が終了した場合は，先に退避した各レジスタを復帰してFARリターンします．

なお，コマンド・パケットのステータス・ワードのセットは，各コマンド処理の中で行われます．また，サポートしていないコマンド・コールに対しては，コマンド・パケットのステータス・ワードの**done**ビットのみをセットしてコマンドの正常終了とします．

⑤ 初期化ルーチン(initialize)

このルーチンでは，RAMディスクの初期化を行います．本RAMディスクは別電源を使用しているため，PC9801の電源を切ったとしても，ディスクの電源が供給されている限りディスク内容の保存が可能な仕様になっています．したがって，一度初期化が実行されていれば，RAMディスク本体の電源を切らない限り初期化を行う必要はありません．

このため，ルーチンでは最初にディスクの内容が保存されているかどうかのチェックを行います．方法は，セクタ0(予備セクタとして使用)の先頭にチェック・

ワード **5AB6H** を書き込んでおき，この値が保存されていれば電源が切られなかったものとして以下の初期化処理は行いません．なお，チェック・ワードの値 **5AB6H** にはとくに意味をもっていません．

イニシャルは，チェック用予備セクタ，FAT，そしてディレクトリについて行います．

まず最初に，セクタ-0 のチェック用予備セクタの先頭にチェック・ワード **5AB6H** を書き込みます．

FAT のイニシャルは，FAT エリアであるセクタ 1～12 を **00H** でクリアします．ただし，FAT の先頭には，3 バイトの **FFH** を書き込みます．MS-DOS 標準ディスクでは FAT-ID が書き込まれていますが，本 RAM ディスクは標準ディスクではないのでとくに必要ではありませんから，初期化時の FAT の先頭 3 バイトには，ダミーとして **FFH** を書き込んでおきます．

ディレクトリのイニシャルは，ディレクトリ・エリアのセクタ 13～20(ディレクトリ 128 個分)を **00H** でクリアします．

最後に，デバイス・ドライバのインストール完了を示すメッセージをコンソールに出力した後で，コマンド・パケットに戻り値をセットします．内容は，ユニット数に 1 を，ブレーク・アドレスには初期化ルーチンの先頭であるイニシャル作業用のバッファ「**rwbuf**」のアドレスを，BPB 配列へのポインタには BPB 配列テーブル「**bpbtable**」のアドレスを，そしてステータス・ワードの **done** ビットに 1 をセットしてリターンします．

なお，処理については図**19**(p.81)を参照してください．

⑥ メディア・チェック・ルーチン(mediacheck)

このルーチンは，RAM ディスクではメディアを交換することがあり得ないためコマンド・パケットの戻り値には常に 1 を，そしてステータス・ワードの **done** ビットに 1 をセットしてリターンします．

なお，処理については図**20**(p.81)を参照してください．

⑦ BPB 作成ルーチン(buildbpb)

このルーチンでは，コマンド・パケットのメディア・ディスクリプタ・バイトに **00H** を，BPB ポインタに BPB テーブル「**bpb**」のアドレスを，そしてステータス・ワードの **done** ビットに 1 をセットしてリターンします．なお，処理については図**21**(p.81)を参照してください．

⑧ ディスク読み出しルーチン(diskread)

このルーチンは，セクタの読み出しを行います．コマンド・パケットからバッファ・アドレス，読み出し開始セクタ番号，読み出しセクタ数を取り出して，まずセクタ番号のチェックを行い 4095 セクタを超えている場合はエラー処理を行います．超えていなければ，サブルーチン「**ramread**」による読み出しを実行します．

サブルーチンによる読み出しは，まず最初にセグメントのラップアラウンドの処理を行います．これは，読み出しセクタ数に 128 以上の値(転送するバイト数が 64 K バイトを超えてしまう)が設定されているような場合に，正しいバッファ・アドレスにデータを転送するために，セグメント：オフセットの値を正規化するものです．

この後で，セクタ番号を RAM ディスクのセクタ番号レジスタに書き込み，そのセクタから 1 セクタ分 512 バイト(256 ワード)のデータを読み取って転送を行います．読み出しは，セクタ番号を＋1 しながらセクタ数分だけ繰り返し実行されます．

最後に，読み出しエラーがない場合には，コマンド・パケットのステータス・ワード **done** ビットに 1 を，エラーがあった場合には **done** ビット，**error** ビットに 1，エラー・コードに 8 (不正なセクタ番号に対するアクセス)をセットしてリターンします．

なお，処理については図**22**(p.82)を参照してください．

⑨ ディスク書き込みルーチン(diskwrite)

このルーチンは，セクタの書き込みを行います．RAM ディスクではベリファイの必要がありませんので，コマンド・コード 8 (ライト)，コマンド・コード 9 (ライト&ベリファイ)ともに同一の処理を行います．構造的には読み出しと書き込みの違いだけで，「読み出しルーチン」とほぼ同じ内容となっています．

コマンド・パケットからバッファ・アドレス，書き込み開始セクタ番号，書き込みセクタ数を取り出し，セクタ番号のチェックを行った後でサブルーチン「**ramwrite**」による書き込みを実行します．

サブルーチンによる書き込みは，読み出しサブルーチン「**ramread**」同様にセグメントのラップアラウンドの処理を行った後で，セクタ番号を RAM ディスクのセクタ番号レジスタに書き込み，そのセクタから 1 セクタ分の 512 バイト(256 ワード)のデータを書き込みます．書き込みは，セクタ番号を＋1 しながらセクタ数分だけ繰り返し実行されます．

最後に，書き込みエラーがない場合には，コマンド・パケットのステータス・ワード **done** ビットに 1 を，エラーがあった場合には **done** ビット，**error** ビットに 1，エラー・コードに 8 (不正なセクタ番号に対するアクセス)をセットしてリターンします．

なお，処理については図**23**(p.82)を参照してください．

デバイス・ドライバのアセンブルと使用方法

リスト 1 に示したアセンブル・リスト **RAMDISK**.

〈図24〉
デバイス・ドライバ
作成の作業例

```
A>DIR RAMDISK.*↵

 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥RAMDISK

RAMDISK  ASM     7152  89-05-06  21:19
       1 個のファイルがあります
  7897088 バイトが使用可能です

A>MASM RAMDISK;↵
The Microsoft MACRO Assembler
Version 1.20, Copyright (C) Microsoft Inc. 1981,82,83


Warning Severe
Errors  Errors
0       0

A>LINK RAMDISK;↵
Microsoft (R) Overlay Linker  Version 3.51
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986. All rights reserved.

Warning: no stack segment

A>EXE2BIN RAMDISK↵

A>DIR RAMDISK.*↵

 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥RAMDISK

RAMDISK  ASM     7152  89-05-06  21:19
RAMDISK  OBJ      955  89-05-07  17:35
RAMDISK  EXE     1557  89-05-07  17:35
RAMDISK  BIN     1045  89-05-07  17:35
       4 個のファイルがあります
  7864320 バイトが使用可能です

A>
```

〈図25〉“CONFIG.SYS”の内容

```
files=10
buffers=10
device=ramdisk.bin
shell=a:¥command.com a:¥ /p
```

LST(生成コードは省略)をもとに，ソース・プログラム **RAMDISK.ASM** を作成してください．コーディングが終了するとアセンブルを行いますが，アセンブル・エラーが発生した場合には，コーディングを再確認してください．エラーがなければ **LINK**，**EXE2BIN** コマンドを使用してデバイス・ドライバ・ファイル **RAMDISK.BIN** を作成します．

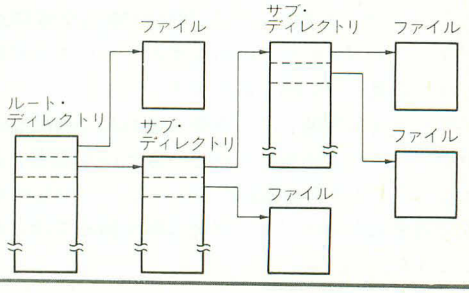
図24にアセンブル，**LINK**，**EXE2BIN** によるデバイス・ドライバ・ファイルの作成例を示します．完成したファイル **RAMDISK.BIN** の大きさは1027バイトになっているはずです．違うようであれば，再度コーディングを確認してください．

デバイス・ドライバが完成したところで，いよいよMS-DOSへの組み込みを行います．まず，**RAMDISK.BIN** をファイル“**CONFIG.SYS**”に登録します．一般的な“**CONFIG.SYS**”の内容を図25に示します．後は“**CONFIG.SYS**”と“**RAMDISK.BIN**”をシステム・ディスクのルート・ディレクトリにコピーしてすべてが完成です．パーソナル・コンピュータのブート・ディスクにフロッピ・ディスクを挿入してシステムの再起動をしてください．

「PC-9801 RAM_DISK Ver0.10 を組み込みます．」

Keyword

ルート・ディレクトリとサブ・ディレクトリ　MS-DOSでは図に示すようにディレクトリが階層構造になっています．この階層構造で最上位に位置するディレクトリをルート・ディレクトリと呼び，これはディスクの決められた位置に一つだけ用意されています．その下にはファイルを定義するのと同じ考え方でいくつでもディレクトリを作成することができます．このようにして作られたディレクトリをサブ・ディレクトリと呼び，ルート・ディレクトリと区別します．



とメッセージが出力されるはずです．これでRAMディスク・デバイス・ドライバがインストールされました．ドライブ番号は，これまで使用していた最後のドライブ番号の次の番号が割り当てられます．図26に使用例を示しますが，表中ではRAMディスクがドライブ“D:”に割り当てられています．

ところで，作成したデバイス・ドライバの試験方法については誌面の都合もあり省略させていただきました．MS-DOSのDEBUGコマンドを使用して行う方法が一般的ですが，通常のアプリケーション・プログラムのデバッグと違い，デバイス・ドライバのデバッグは多少面倒です．今回作成したデバイス・ドライバは，アセンブラ・ソースで数百行ですし，構造的にもさほど難しい部分はないと思いますので，回路図とリストを参考にしてチャレンジして見てください．いずれ機会があれば紹介したいと考えています．

〈図26〉MS-DOSの起動とRAMディスクの使用例

```
NEC PC-9801 Series Personal Computer

ﾏｲｸﾛｿﾌﾄ MS-DOS ﾊﾞｰｼﾞｮﾝ 2.11
Copyright 1981,82,83 Microsoft Corp. / NEC Corporation


PC-9801 RAM_DISK Ver 0.10 を組み込みます

Command ﾊﾞｰｼﾞｮﾝ 2.11
現在の日付は 1989-05-07 (月) です
日付を入力してください :↵
現在の時刻は 17:42:24:00 です
時刻を入力してください :↵


A>D:↵


D>COPY A:¥RAMDISK¥RAMDISK.*↵


D>DIR↵
 ドライブ D: のディスクにはボリュームラベルがありません
 ディレクトリは D:¥

RAMDISK  ASM     7152  89-05-06  21:19
RAMDISK  OBJ      955  89-05-07  17:35
RAMDISK  EXE     1557  89-05-07  17:35
RAMDISK  BIN     1045  89-05-07  17:35
       4 個のファイルがあります
 2074624 バイトが使用可能です


D CHKDSK↵


 2086400 バイト : 全ディスク容量
   11776 バイト : 4 個のユーザーファイル
 2074624 バイト : 使用可能ディスク容量

  655360 バイト : 全メモリ
  491712 バイト : 使用可能メモリ

D>
```

● 最後に

MS-DOSは，16ビット・パーソナル・コンピュータの標準的なOSとして使用されています．そして，そのもとで使用することのできるアプリケーション・プログラムは数え切れないほどの膨大な数になっています．同様に，プログラム開発環境についてもFortran, Pascal, Cなどとほとんどの高級言語が使用可能となっています．

高級言語を使用してプログラム開発を行うことのメリットは，開発効率が良い，構造的なプログラミングによりプログラムの見通しが良いなど多くの利点をもっています．

しかし，フロッピ・ディスク・ベースでの開発では，コンパイル処理などに費やされる時間は高速だといわれるC言語でも数千行の処理には相当な時間が必要となります．Pascalでは同程度のコンパイル処理に数十分を必要とした例もあります．

様々な技術現場では，時間に追われた過酷な労働の中でプログラム開発を行っているという現実もあり，RAMディスクの高速動作を生かして，少しでも高品位で効率的なプログラム開発に取り組んではいかがでしょうか．

◆ 参考・引用* 文献 ◆

(1) 標準MS-DOSハンドブック，アスキー．
(2) MS-DOSプログラマーズハンドブック，アスキー．
(3) MS-DOSユーザーズマニュアル，日本電気．
(4) PC-9801シリーズ・テクニカルデータブック，アスキー．
(5) PC-9801VXハードウェアマニュアル，日本電気．
(6)* MCC-1取扱説明書，山下電気．
(7) 半導体技術資料No 14, 47，テキサスインスツルメンツ．
(8)* TC511000技術資料，東芝．
(9)* EXO-3クリスタルオシレータ技術資料，KSS．
(10) 金井英和他；特集 MS-DOSの基礎と応用，インターフェース，1985年1月号．
(11) 金井英和他；特集 MS-DOSの徹底活用，インターフェース，1985年2月号．
(12) 保存版特集 MS-DOS活用テクノロジー，マイコンピュータ，1986年No.20．

〈リスト 1〉 RAM ディスク・ドライバのアセンブル・リスト　RAMDISK.LST

```
page    63,130
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;                                                         ;
        ;       PC9801 RAM ディスク・デバイス・ドライバ            ;
        ;                                                         ;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DATAIN  equ     80dch           ;データ読み出しポート番号
DATAOUT equ     80dch           ;データ書き込みポート番号
SECREG  equ     80deh           ;セクタ番号レジスタ・ポート番号
CHECK   equ     5ab6h           ;イニシャル済みチェック・ワード

cseg    segment
        assume  cs:cseg, ds:cseg, es:cseg, ss:cseg

        ;
        ;  デバイス・ヘッダの定義
        ;
header          dd      -1              ;リンク情報（初期値は-1にセットする）
                dw      2000h           ;デバイス属性．NON-IBM ブロック型デバイス
                dw      strategy        ;ストラテジ・エントリ・ポイント
                dw      interrupt       ;割り込みエントリ・ポイント
                db      1               ;ユニット数 =1
                db      7 dup (' ')     ;ダミー

        ;
        ;  RAM ディスク BPB の定義
        ;
bpbtable        dw      offset bpb      ;BPB 配列ポインタ・テーブル

bpb             dw      512             ; 1セクタのバイト数
                db      1               ; 1アロケーション当たりのセクタ数
                dw      1               ;予備のセクタ数
                db      1               ;ＦＡＴの数
dirnum          dw      128             ;ルート・ディレクトリの数
maxsec          dw      4096            ;論理的な総セクタ数
devdis          db      00h             ;メディア・ディスクリプタ
fatnum          dw      12              ; 1ＦＡＴ当たりのセクタ数
packet          dd      ?               ;コマンド・パケット・アドレス格納エリア

        ;
        ;  コマンド・ジャンプ・テーブル
        ;
jmptable        dw      initialize      ; 0- 初期化コマンド
                dw      mediacheck      ; 1- メディア・チェック・コマンド
                dw      buildbpb        ; 2- BPB 作成コマンド
                dw      exit            ; 3- (Not used)
                dw      diskread        ; 4- ディスク読み出しコマンド
                dw      exit            ; 5- (Not used)
                dw      exit            ; 6- (Not used)
                dw      exit            ; 7- (Not used)
                dw      diskwrite       ; 8- ディスク書き込みコマンド
                dw      diskwrite       ; 9- ディスク書き込みコマンド (Verify)
                dw      exit            ;10- (Not used)
                dw      exit            ;11- (Not used)
                dw      exit            ;12- (Not used)

proca   proc    far
```

```
        ;-----------------------------------------------
        ;  ストラテジ・エントリ・ルーチン
        ;-----------------------------------------------
strategy:
        mov     word ptr cs:[packet],bx         ;コマンド・パケットのアドレス ES:BXを
        mov     word ptr cs:[packet+2],es       ;  [packet]に格納する
        ret                                     ;FAR リターンで戻る

        ;-----------------------------------------------
        ;  割り込みエントリ・ルーチン
        ;-----------------------------------------------
interrupt:
        push    ax                              ;全レジスタの退避
        push    bx
        push    cx
        push    dx
        push    ds
        push    es
        push    di
        push    si

        lds     di,dword ptr cs:[packet]        ;DS:DI にコマンド・パケット・アドレス
        mov     bl,byte ptr [di+2]              ;BLレジスタにコマンド番号
        cmp     bl,13                           ;コマンド番号エラーか？
        jnb     _cmderr
        mov     bh,0
        add     bx,bx
        call    cs:word ptr jmptable [bx]       ;コマンド処理ルーチンをコールする

_intret:
        pop     si                              ;全レジスタの復帰
        pop     di
        pop     es
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret                                     ;FAR リターンで戻る

_cmderr:
        mov     word ptr [di+3],8103h           ;コマンド・エラー・ステータスをセット
        jmp     _intret
proca   endp

        ;-----------------------------------------------
        ;  コマンド正常終了ルーチン
        ;-----------------------------------------------
exit:
        mov     word ptr [di+3],0100h           ;コマンド正常終了ステータスをセット
        ret

        ;-----------------------------------------------
        ;  メディア・チェック・コマンド処理
        ;-----------------------------------------------
mediacheck:
        mov     byte ptr [di+14],1      ;メディア変更なし(1)をパラメータ領域にセット
        jmp     exit
```

〈リスト 1〉RAM ディスク・ドライバのアセンブル・リスト　RAMDISK.LST(つづき)

```
        ;-----------------------------------------
        ;  BPB の作成コマンド処理
        ;-----------------------------------------
buildbpb:
        mov     byte ptr [di+13],00h            ;メディア・ディスクリプタとBPBのアドレス
        mov     word ptr [di+18],offset bpb     ;  をパラメータ領域にセット
        mov     word ptr [di+20],cs
        jmp     exit

        ;-----------------------------------------
        ;  ディスク書き込みコマンド処理
        ;-----------------------------------------
diskwrite:
        mov     bx,word ptr [di+18]             ;BX= 書き込みセクタ数
        mov     dx,word ptr [di+20]             ;DX= 書き込み開始セクタ番号
        lds     si,dword ptr [di+14]            ;DS:SI= バッファ・アドレス
        mov     ax,dx
        add     ax,bx
        cmp     ax,cs:[maxsec]                  ;最大セクタ数を超えていないかを
        jnb     _rwerr                          ;  チェックする
        call    ramwrite                        ;書き込みルーチンをコール
        lds     di,dword ptr cs:[packet]        ;コマンド・パケットのアドレスをゲットする
        jmp     exit

        ;-----------------------------------------
        ;  ディスク読み出しコマンド処理
        ;-----------------------------------------
diskread:
        mov     bx,word ptr [di+18]             ;BX= 読み出しセクタ長
        mov     dx,word ptr [di+20]             ;DX= 読み出し開始セクタ番号
        les     di,dword ptr [di+14]            ;ES:DI= バッファ・アドレス
        mov     ax,dx
        add     ax,bx
        cmp     ax,cs:[maxsec]                  ;最大セクタ数を超えていないか
        jnb     _rwerr                          ;  チェックする
        call    ramread                         ;読み出しルーチンをコール
        lds     di,dword ptr cs:[packet]        ;コマンド・パケットのアドレスをゲットする
        jmp     exit
_rwerr:
        lds     di,dword ptr cs:[packet]        ;コマンド・パケットのアドレスをゲットする
        mov     word ptr [di+18],0
        mov     word ptr [di+3],8108h           ;読み書きコマンドの異常終了ステータスをセット
        ret

        ;-----------------------------------------
        ;  ディスク書き込みルーチン
        ;
        ;  Entry
        ;    DS:SI : バッファ・アドレス
        ;    DX    : 開始セクタ番号
        ;    BX    : 書き込みセクタ数
        ;-----------------------------------------
ramwrite:
        mov     ax,si           ;ラップアラウンドを解消するため DS:SIを正規化する
        and     si,000fh
        mov     cl,4
        shr     ax,cl
        mov     cx,ax
        mov     ax,ds
        add     ax,cx
        mov     ds,ax

        mov     ax,dx
        push    dx
        mov     dx,SECREG       ;DX= セクタ番号レジスタのポート番号
        out     dx,ax           ;セクタ番号レジスタに書き込み開始セクタ番号をセット
        mov     cx,256          ;CX= 物理的な 1セクタ長 (256ワード)
        mov     dx,DATAOUT      ;DX= データ書き込みポート番号
        cld

_wrl:
        lodsw                   ;DS:SI で示されるバッファから
        out     dx,ax           ;  1セクタ分のデータを転送して書き込む
        loop    _wrl

        pop     dx
        inc     dx              ;書き込みセクタ番号を+1する
        dec     bx              ;書き込みセクタ数を-1する
        jnz     ramwrite        ;書き込みセクタ数がゼロになるまで繰り返す
        ret

        ;-----------------------------------------
        ;  ディスク読み出しルーチン
        ;
        ;  Entry
        ;    ES:DI : バッファ・アドレス
        ;    DX    : 開始セクタ番号
        ;    BX    : 読み出しセクタ数
        ;-----------------------------------------
ramread:
        mov     ax,di           ;ラップアラウンドを解消するため ES:DIを正規化する
        and     di,000fh
        mov     cl,4
        shr     ax,cl
        mov     cx,ax
        mov     ax,es
        add     ax,cx
        mov     es,ax

        mov     ax,dx
        push    dx
        mov     dx,SECREG       ;DX= セクタ番号レジスタのポート番号
        out     dx,ax           ;セクタ番号レジスタに読み出し開始セクタ番号をセット
        mov     cx,256          ;CX= 物理的な 1セクタ長(256ワード)
        mov     dx,DATAIN       ;DX= データ読み出しポート番号
        cld
_rdl:
        in      ax,dx           ;ES:DI で示されるバッファへ
        stosw                   ;  1セクタ分のデータを転送する
        loop    _rdl

        pop     dx
        inc     dx              ;読み出しセクタ番号を+1する
        dec     bx              ;読み出しセクタ数を-1する
        jnz     ramread         ;読み出しセクタ数がゼロになるまで繰り返す
        ret
```

```
        ;-----------------------------------------------
        ;  初期化コマンド処理
        ;    デバイス・ドライバの組み込み時に一度だけコールされる。
        ;    初期化終了以降に以下のプログラムはメモリから削除される。
        ;-----------------------------------------------
rwbuf   db      512  dup (0)             ;初期化ルーチンで使用する
                                        ;  読み書き用バッファ


initialize:
        push    cs
        pop     es
        mov     di,offset rwbuf         ;ES:DI に rwbuf のアドレス
        mov     dx,0
        mov     bx,1
        call    ramread                 ;予備セクタを読み出す
        mov     ax,word ptr cs:[rwbuf]
        cmp     ax,CHECK                ;チェック・ワードか書かれているかテストする
        jz      _init                   ;書かれていれば初期化を行わない
        call    _diskinit               ;書かれていなければ初期化を行う
_init:
        mov     dx,offset msg
        call    msgout                  ;RAM ディスクのインストール・メッセージ出力
        lds     di,dword ptr cs:[packet]        ;コマンド・パケット・パラメータ領域に
        mov     byte ptr [di+13],1              ;  ユニット数=1をセット
        mov     word ptr [di+14],offset rwbuf
        mov     word ptr [di+16],cs             ;  ブレーク・アドレスをセット
        mov     word ptr [di+18],offset bpbtable
        mov     word ptr [di+20],cs             ;  BPB 配列へのポインタをセット
        mov     word ptr [di+3],0100h           ;コマンド正常終了ステータスをセット
        ret

        ;-----------------------------------------
        ;  FAT とディレクトリの初期化
        ;-----------------------------------------
_diskinit:
        mov     ax,cs                           ;rwbuf をゼロ・クリアする
        mov     ds,ax
        mov     es,ax
        mov     di,offset rwbuf
        mov     cx,256
        mov     ax,0
        cld
        rep     stosw

        mov     word ptr cs:[rwbuf+0],CHECK     ;rwbuf の先頭にチェック・ワードをセット
        mov     dx,0
        mov     bx,1
        mov     si,offset rwbuf
        mov     ax,cs
        mov     ds,ax
        call    ramwrite                        ;予備セクタを書き込む（セクタ0）

        mov     word ptr cs:[rwbuf+0],0ffffh    ;rwbuf の先頭にダミー・バイトをセット
        mov     word ptr cs:[rwbuf+2],00ffh
        mov     dx,1
        mov     bx,1
        mov     si,offset rwbuf
        mov     ax,cs
        mov     ds,ax
        call    ramwrite                        ;FAT の先頭セクタを書き込む（セクタ1）

        mov     word ptr cs:[rwbuf+0],0         ;rwbuf をゼロでクリアする
        mov     word ptr cs:[rwbuf+2],0
        mov     cx,19
_dini0:                                         ;全ての FAT領域をゼロでクリアする
        push    cx                              ;  （セクタ2～セクタ12）
        mov     bx,1                            ;全てのディレクトリ領域をゼロでクリアする
        mov     si,offset rwbuf                 ;  （セクタ13～セクタ20）
        mov     ax,cs
        mov     ds,ax
        call    ramwrite                ;FAT の第２セクタからディレクトリの最終セクタ
        pop     cx                      ;  までを書き込む（セクタ2～セクタ20）
        loop    _dini0
        ret

msgout:
        push    cs
        pop     ds
        mov     ah,9
        int     21h
        ret

        msg     db      13,10,'PC-9801 RAM_DISK Ver 0.10 を組み込みます',13,10,'$'




cseg    ends
        end
```

第3章 時計表示と縮小印字ドライバを作る
キャラクタ型と常駐型デバイス・ドライバの作り方

プリンタやコンソールなどは1文字単位の入出力が行われるキャラクタ型デバイス・ドライバです．デバイス・ドライバの構造を理解するために「何もしない」プログラムから始め，時計表示や縮小印字を行う例などを紹介します．

石垣 武彦

MS-DOSで使われる入出力装置には，さまざまな種類のものがあります．昔コンピュータを習った方には『入出力装置』と言ったほうが通りがいいでしょうが，最近は『デバイス』と呼ぶことのほうが多くなっています．この『デバイス』の種類には，ざっとあげるだけでも，スクリーン，キーボード，プリンタ，フロッピ・ディスク，ハード・ディスク，CD-ROMなどさまざまなものがあります．MS-DOSではとくに，スクリーン・デバイスとキーボード・デバイスとの二つを組み合わせてコンソール・デバイスと呼びます．

それぞれのデバイスは，さまざまな特性や独自のインターフェースをもっています．特性とは，たとえばスクリーン・デバイスの表示文字数やプリンタの制御コマンド，ディスクの記憶容量などがあげられます．接続インターフェースでは，キーボードがシリアル接続かパラレル接続なのか，ディスク装置はSCSIなのかSASIなのかといったようなものです．

● デバイス・ドライバによってユーザはこまかいことを気にしなくてよい

このようにさまざまな種類のデバイスの違いを吸収して，MS-DOS上からは同一に見えるような役割をはたすのがデバイス・ドライバです．デバイス・ドライバを通すことによって，ユーザはそのディスクがSCSIであるかSASIかどちらのインターフェースでつながっているかなどを知らなくともいいし，またスクリーン・デバイス・ドライバに1文字を送ったり，キーボード・デバイス・ドライバから1文字を読み込んだりするという単純操作だけになります．こうしてユーザは，わずらわしい作業から解放されるのです(図1)．

このようなMS-DOSのデバイス・ドライバには，大きく分けて，キャラクタ型とブロック型の二つのタイプがあります．キャラクタ型デバイス・ドライバは，1バイト(キャラクタ)単位でデータを扱うデバイス・ドライバであり，ブロック型デバイス・ドライバは，複数バイトのブロック単位でデータを扱うデバイス・ドライバです．

フロッピ・ディスクやハード・ディスクは，すべてブロック型デバイスといえます．そしてディスク装置以外のほとんどのデバイスはキャラクタ型です．

この章では，キャラクタ型デバイス・ドライバを「標準型」と「特殊型」の二つのタイプに分けて説明することにします．本来デバイス・ドライバにはこのような分類はありません．この二つのタイプの違いというのは，MS-DOSが頻繁に使用するデバイス・ドライバであるかどうかという違いだけです．「標準デバイス・ドライバ」は，MS-DOSがオペレータと対話するためにはなくてはならないものです．それに対して，「特殊デバイス・ドライバ」は必須ではありませんし，多くの場合，デバイス・ドライバの型式をとらなくても実現できます．

本章では，キャラクタ型デバイス・ドライバのことを，たんにデバイス・ドライバと呼ぶことがあるので注意してください．ブロック型については第2章を参照してください．

標準キャラクタ型デバイス・ドライバ

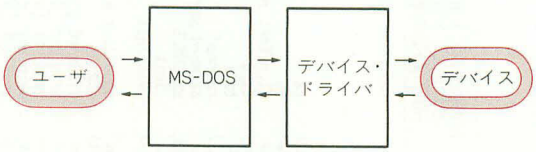
標準デバイス・ドライバは，

NULデバイス
CONデバイス
CLOCKデバイス

の三つの種類のデバイスを扱うドライバです．この三つのデバイスは，MS-DOSオペレーションでもっとも重要な役割をはたします．そのため，MS-DOSがメーカから供給される際には必ず組み込まれています．

これらのデバイス・ドライバは，システムの中にそ

〈図1〉デバイスとユーザ



ユーザからはデバイスのこまかい仕様はみえていない

れぞれ一つしか存在しません．仮に，ユーザが新しいCONデバイス・ドライバを作成したとすると，以前のCONデバイス・ドライバは機能しなくなり，すべての機能を新しいCONデバイス・ドライバにゆだねるようになります．

● NULデバイスは何もしない

NULというデバイスは，何もしない論理出力デバイスです．実際の外部装置に接続されているわけではありません．またデバイス・ドライバ・チェーンの先頭マーカとしての役割もはたしています(図2)．**CONFIG.SYS**などで記述された組み込み以外のデバイス・ドライバは，このNULデバイス・ドライバの後にチェーンされていきます．

ダミーの出力デバイスであるNULは，ユーザから見た場合，それほど有用なデバイスではありませんが，

```
A>COPY A:*.*  B: >NUL
```

などのようにして，アプリケーションの出力をNULに向けることで，余計なメッセージを抑制するような使い方をされることがあります．また，MS-DOSのデバイス・ドライバのオーバヘッドを計測するような使い方も考えられます．

NULデバイスは出力デバイスなので，

```
A>TYPE NUL
```

などのように入力デバイスとして使用しても，何もせずに終了します．これはNULデバイス・ドライバからエラーが返されるからです．

● CONデバイスは入出力の二刀流

CONデバイスはいわゆるコンソール・デバイスであり，標準入力デバイスと標準出力デバイスの両方の機能をもっています．一般に標準入力デバイスはキーボードであり，標準出力デバイスはスクリーンです．したがって，CONデバイス・ドライバの作成は注意を要します．なぜなら，CONデバイスの処理能力がシステム全体のスループットを決めてしまうからです．

たとえば，画面表示出力をたんなるダム・ターミナル(指定された文字を加工せずに出力するだけ)としてしまえば簡単ですが，一般にエスケープ・シーケンスと呼ばれる画面制御機能をもたせて作成する場合があります．カーソル移動やスクロール制御などの機能が，その代表的なものといえましょう．

英数字のみを扱う外国のMS-DOSマシンはエスケープ・シーケンス機能の作成が容易で，そのため画面出力も高速に行われます．それ対して，漢字を考慮した国産マシンの表示速度は比較的低速です．

表示を高速にするために，最近はこのような低速のMS-DOS標準出力を使わずに，自ら画面出力ルーチンをもったアプリケーションが多くなりました．しかし，このために犠牲にされるのはポータビリティ(移植性)です．これはMS-DOSマシンに限らず，OSメーカとユーザとのいたちごっこです．OSメーカは常にポータビリティを追求し，ユーザは高速性を追求するのです．OS/2にせよMacintoshにせよ，高速性を追求するユーザは，いつもOSの裏をかこうとしています．

CONデバイスは，入出力の両方の機能をもったデバイスなので，

```
A>COPY CON AUTOEXEC.BAT
```

とすれば，キーボード入力が**AUTOEXEC.BAT**というファイルに反映されます．エディタが手元にない場合などによく使用されるテクニックです．

また，出力デバイスとして，

```
A>COPY AUTOEXEC.BAT CON
```

とすることもできます．これは**AUTOEXEC.BAT**ファイルの内容をスクリーンに表示させます．TYPEコマンドでは1回に一つのファイルしか表示できませんが，

```
A>COPY *.* CON
```

などとすれば，カレント・ディレクトリのすべてのファイルの内容を表示させることができることになります．

● CLOCKデバイスはユーザが直接操作するわけではない

CLOCKデバイスは，MS-DOS内部で使用されるデバイスで，ファイルの作成日付けなどをMS-DOSが付加する際に呼び出されます．したがってデバイスとは呼ばれていますが，ユーザが直接操作したりすることはあまり行われません．したがって，

```
A>TYPE CLOCK
```

というのは，画面に現在時刻を表示するわけではありませんし，

```
A>COPY AUTOEXEC.BAT CLOCK
```

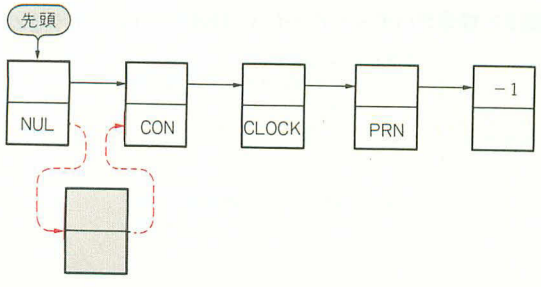
などとすれば，MS-DOSのシステム時間がおかしなものになってしまう可能性があります．

● 準標準のデバイス・ドライバ

以上の三つのキャラクタ・デバイスは，MS-DOSに必須のものでした．しかし，これら以外にも「準標準」と見なせるデバイス・ドライバが存在します．それらには，

▶ PRNデバイス

〈図2〉デバイス・ドライバ・チェーン



▶ AUX デバイス

などがあります．ただし，これらのデバイス・ドライバが存在しなくてもMS-DOSは動作します．したがって，MS-DOSがメーカから供給される場合に必ず添付されるものとは限りません．

それぞれ，PRNデバイスはプリンタを，AUXデバイスはシリアル回線を制御するデバイス・ドライバです．キャラクタ・デバイスの名称は，システムの中で唯一ひとつしか存在できないので，複数のプリンタが接続できるシステムでは**LST1**デバイス，**LST2**デバイスのようにデバイス・ドライバが別々の名称をもっていることもあります．

これらのデバイス・ドライバが「準標準」である理由は，たんに標準キャラクタ・デバイスと同じようなMS-DOSオペレーションが可能であるということです．つまりMS-DOSにとって必須ではないが，入出力のためのデバイスとして機能するということです．たとえば，

```
A>COPY AUTOEXEC.BAT PRN
```

とすれば，ファイルの内容がプリンタに出力されますし，

```
A>COPY AUTOEXEC.BAT AUX
```

とすれば，ファイルの内容がシリアル回線へ出力されます．

ユーザにとって，それらが標準デバイスであるか準標準デバイスであるかの区別はできません．

特殊キャラクタ型デバイス・ドライバ

特殊デバイス・ドライバは，いままで説明してきたデバイス・ドライバとはおもむきを変えます．このデバイス・ドライバは，MS-DOSのメモリ中に常駐するだけのために，その型式をとっているにすぎないことが多いのです．もしくは，どのドライバよりも最初に起動をしたい場合にも，特殊デバイス・ドライバの型式をとることがあるでしょう．またデバイス・ドライバの型式をとることによって，MS-DOSのI/Oコントロール機能が使用できるという利点もみのがせません．

どの理由にせよ，入出力デバイスとしての機能を持ち合わせていないようなデバイス・ドライバを，特殊デバイス・ドライバとして分類しましょう．

したがって，特殊デバイス・ドライバは，MS-DOSのリダイレクション機能を使用してデバイスとの入出力を行うことは一般にはできません．多くの場合，デバイス・ドライバがエラーを返すことで問題を生じませんが，そうでない可能性がなくなるということではありません．

こうした意味で，先ほど出てきたCLOCKデバイス・ドライバは特殊型に分類されなければなりませんが，デバイス・ヘッダで指定することによって，例外としてMS-DOSに認識されています(図3)．

● MS-DOS 3.3には特殊デバイス・ドライバが二つ付属している

特殊型を代表するデバイス・ドライバとしては，PC9801シリーズのMS-DOS 3.3に付属する，

▶グラフィック・デバイス・ドライバ
(**GRAPH $デバイス**)

▶ EMSデバイス・ドライバ
(**EMMXXXX0デバイス**)

などがあげられます．

グラフィック・デバイス・ドライバは，メモリへ常駐するだけの目的のためにデバイス・ドライバの型式をとっています．したがってデバイス・ドライバではなく，常駐型アプリケーション(TSR-Terminate but Stay Regident)の型式をとってもかまわないのです．

グラフィック・デバイス・ドライバがメモリに常駐した後，ユーザは割り込みおよびFAR CALLを使用してグラフィック・ファンクションを実行します．

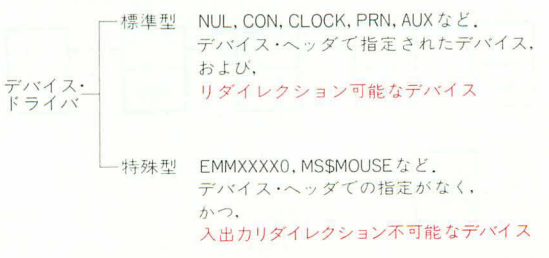
EMSデバイス・ドライバは，誰よりも先にメモリへ常駐する目的でデバイス・ドライバの型式をとっています．TSRで実現することも可能ですが，そうするとほかのデバイス・ドライバがEMSを利用することができなくなってしまいます．

また，このデバイス・ドライバは，EMSマネージャがアクティブであるかどうかを判断するためにデバイス名を利用しています．アプリケーションが"**EMMXXXX0**"というデバイスをオープンすることができれば，アクティブであると判断するのです．しかし実際の入出力は，**EMMXXXX0**デバイスを通して行うことができないので，注意が必要です．EMSマネージャは，**INT 67H**という割り込みを通して制御するようになっています．詳しくは第4章を参照してください．

● マウスの制御のドライバも特殊型

このほかにも，マウスを制御するためのマウス・デバイス・ドライバが存在します．このデバイス・ドライバも物理デバイスを制御しているにもかかわらず，MS-DOS自身がマウスに関するシステム・ファンクシ

〈図3〉標準デバイス・ドライバと特殊デバイス・ドライバ



ョン・コールをもっていないために，特殊型に属します．実際，**MOUSE.SYS** というファイル名のデバイス・ドライバと **MOUSE.COM** というファイル名の TSR が存在します．どちらもマウス・ファンクションを **INT 33H** で呼び出して使用し，まったく同じ動作をするのです．

TSR とキャラクタ型デバイス・ドライバ

デバイス・ドライバは，TSR として動作させることが可能なことは前に述べました．しかも MS-DOS に限っていえば，TSR を作成するよりもデバイス・ドライバを作成するほうがはるかに容易です．

それというのも，TSR の作成はまったく作成者の技量にゆだねられており，一定の作成方法が確立されていないからです．デバイス・ドライバは図 4 のように，構造が MS-DOS の規格によって決められていて，迷うことは少ないのです．詳しい規格についてはこのあとで述べます．

● TSR とデバイス・ドライバの大きな違いは MS-DOS がその存在を知っているか否か

TSR とキャラクタ型デバイス・ドライバのもっとも大きな違いは，常駐されたもしくは組み込まれたプログラムを，MS-DOS が知っているかどうかということです．デバイス・ドライバの場合には，MS-DOS は規格化されたデバイス・ドライバ・ヘッダによってその内容を把握しています．しかし，TSR の場合には規格化された作成方法は存在しません．したがって，TSR に関して MS-DOS は何も知りません．知る方法がないのです．

たとえば，組み込まれたデバイス・ドライバはそれぞれ 8 文字のデバイス名をもっていますが，TSR はそのような情報をもっていません．

この情報の有無の差は，必要なプログラムがすでにメモリ上に常駐しているかどうかを判断する際に現れます．デバイス・ドライバの場合は，たんにそのデバイス名を MS-DOS のシステム・コールを使用してオープンしてみればよいのです．オープンに成功すれば常駐していることを意味します．それに対して，TSR の場合にはそのような簡便な手段は用意されておらず，なんらかの手段を考えなければなりません．

さらに，プログラムを外部から制御したい場合にも，デバイス・ドライバ型式のほうが便利です．なぜならば，デバイス・ドライバには，IOCTL(I/O コントロール) という機能が MS-DOS によってサポートされていて，デバイス・ドライバ自身をシステム・ファンクション・コールを通して直接制御することができるからです．

● デバイス・ドライバの常駐とその解除

しかし，デバイス・ドライバにも欠点はあります．一つに自由な取り外しが行えないことです．一度常駐したら最後，コンピュータをリセットするまで常駐しています．

TSR は作り方にもよりますが，常駐を解除することができます．必要なときに常駐させ，必要でなくなったらメモリから解放させることは，資源の有効利用につながります．もちろん，TSR の常駐と解放は作成者の責任において行わなければなりません．

取り外しと同様に，組み込む際にもデバイス・ドライバは制約があります．すなわちデバイス・ドライバの組み込みは，MS-DOS の起動時にしか行われないということです．

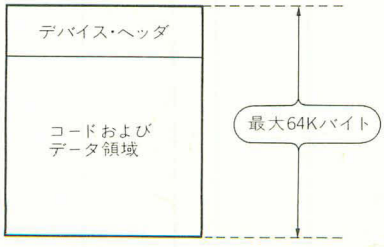
なお，このデバイス・ドライバの組み込みと切り離しを自由に行うユーティリティがいくつか発表されるようになったので，これに関しては改善の方向に向かっているといえるでしょう．

デバイス・ドライバ・ヘッダ

標準型，特殊型を問わず，キャラクタ型デバイス・ドライバの作成方法は同じです．デバイス・ドライバにとって最も重要なのは，デバイス・ヘッダと呼ばれる部分です．デバイス・ヘッダはわずか 18 バイトしかありませんが，ここに記述される情報によって MS-DOS がデバイス・ドライバの種類を判別し，処理を行います．表 1 にデバイス・ヘッダの構造を示します．

デバイス・ヘッダの最初のフィールドは，『次のデバイスへのポインタ』です．言い換えれば，『次のデバイ

〈図 4〉基本的なデバイス・ドライバの構成



〈表 1〉デバイス・ヘッダの構成

2 ワード (4 バイト)	次のデバイスへのポインタ
1 ワード (2 バイト)	デバイスの属性
1 ワード (2 バイト)	デバイス・ストラテジ・エントリ・ポインタ
1 ワード (2 バイト)	デバイス割り込みエントリ・ポイント
8 バイト	デバイス名

ス・ドライバのアドレス』と同じことを意味しています．これは，前にも説明したデバイス・チェーンを形成するために必ず必要なフィールドともいえます．

一つのデバイス・ドライバが一つのデバイスしかサポートしていない場合は，普通 **OFFFFH**（**−1**）を指定しておきます．こうしておくことで，MS-DOS がデバイス・ドライバをメモリ中にローディングする際に自動的にデバイス・チェーンを形成し，正しい値に変更してくれます．

● 複数のデバイスをサポートする場合

問題なのは，一つのデバイス・ドライバが複数のデバイスをサポートしている場合です．この場合には，デバイス・ヘッダも複数必要になります（図 5）．この図では，デバイス・ヘッダを二つもっているデバイス・ドライバを例としてあげています．

『次のデバイスへのポインタ』は，デバイス・ドライバの作成時にはオフセット値のみしか決定できません．セグメント値は，デバイス・ドライバが MS-DOS へ読み込まれる際にしかわからないのです．そのためこのような場合は，デバイス・ドライバの初期化ルーチンが呼び出されたときに設定するようにします．詳しいことは後で説明することにしましょう．

● キャラクタ型におけるデバイスの属性

『デバイスの属性』は 16 ビット（2 バイト）から構成されるフラグで，デバイス・ドライバがどのタイプのものであるかを指定します．図 6 では，ブロック型に関係するビット・フラグの説明は省いてあります．キャラクタ型デバイス・ドライバでは，最上位ビットは必ず 1 でなければなりません．ゼロの場合はブロック型デバイス・ドライバであることを意味するからです．

最下位の 4 ビットは，この章の前半で述べた標準型デバイス・ドライバを指定するフラグです．STI フラグと STO フラグだけは同時に指定することはできますが，ほかのフラグは基本的に排他的ですので，NUL フラグと CLOCK フラグとを同時に指定するようなことは許されていません．たとえば，

STI フラグのみを指定して，KBD デバイス
STO フラグのみを指定して，SCREEN デバイス
STI と STO フラグを指定して，CON デバイス
CLOCK フラグを指定して，CLOCK デバイス
NUL フラグを指定して，NUL デバイス

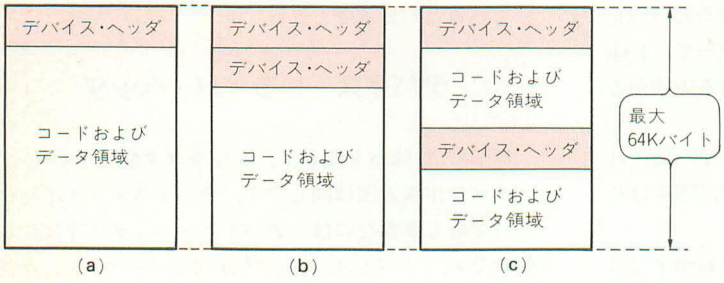
というようにすることが考えられます．デバイスの名称は規定されているわけではありませんので，作成者が自由につけることができます．

● スペシャル・デバイスの指定では画面への文字出力が高速化される

SPCL フラグが指定されたデバイス・ドライバに対しては，MS-DOS は画面への文字出力シーケンスを通常のデバイス・ドライバ型式をとりません．通常のデバイス・ドライバ型式というのは，「出力文字をパケットに入れ，ストラテジ・ルーチンを呼び出し，その後エントリ・ルーチンをコールする」といった手間のかかる呼び出し型式のことをいいます．この方法は OS/2 でも採用され，マルチタスクに向いた呼び出し型式です．

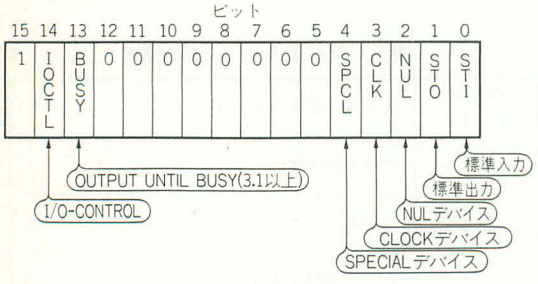
しかし，MS-DOS のようなシングル・タスクの OS ではあまり大きなメリットがありません．逆に，かえ

〈図 5〉さまざまなデバイス・ドライバの構成

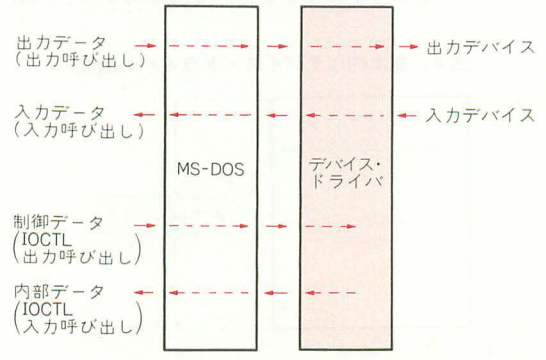


(a)は，ただ一つのデバイスのみをサポートしている基本的なデバイス・ドライバ．
(b)は，二つのデバイスをサポートしたデバイス・ドライバ．デバイス・ヘッダをファイルの先頭にまとめて配置してある．
(c)は(b)と同じだが，デバイス・ヘッダの配置が異なる．

〈図 6〉キャラクタ型デバイスの属性



〈図 7〉I/O コントロール(IOCTL)機能



ってデータ出力速度が遅くなってしまいます．SPCLフラグを1にしておくことで，MS-DOSはALレジスタに出力文字を入れ，**INT 29H**を呼び出すような方式をとるようになります．

● **IOCTLフラグはユーザしだい**

IOCTLフラグは，ユーザがデバイス・ドライバを制御する場合に使われます(**図7**)．アプリケーションがMS-DOSの普通の入出力コールを行うと，データはそのままデバイスとやりとりされます．しかし，IOCTLフラグがonのデバイス・ドライバは，MS-DOSのI/Oコントロール・コールが行えるようになります．

これによって，デバイス・ドライバの内部状態を変更したり，バージョン番号を読み出したりすることができるようになります．

● **ストラテジと割り込みのエントリ・ポイントはオフセットの値だけ**

『デバイス・ストラテジ・エントリ・ポイント』と『デバイス割り込みエントリ・ポイント』の二つは，SPCLフラグがonでないデバイス・ドライバにとっては重要です．この二つのオフセット値を使用して，デバイス・ドライバの呼び出しが行われるからです．

どちらもセグメント値を設定することができないので，ヘッダよりも64Kバイト以内にコードを配置しなければならないことに注意してください．

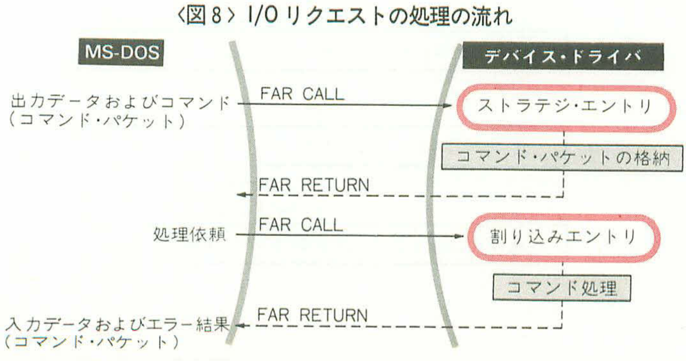
● **デバイスの名前はかってにつけられるが**

最後の『デバイス名』はそれほど重要ではありません．作成者が自由に名前をつけることができます．だからといって，安易に“**TEST**”などという名称の名前をつけないようにしてください．もしこのような名前のデバイス名をつけたら最後，**TEST**というディスク・ファイル名は使用できなくなります．

A>COPY AUTOEXEC.BAT TEST

というオペレーションは**TEST**というファイルではなく，**TEST**というデバイスに対して行われるからです．一般的な規則としては，「使われそうもない名前をつける」ということです．たとえば，“**$TEST$**”というデバイス名ならば，ファイル名と同じになってしまう危険性は少なくなります．

〈図8〉 I/Oリクエストの処理の流れ



〈リスト1〉 デバイス・ヘッダのコーディング例

```
device_header   dw      -1, -1
                dw      8013h
                dw      offset strategy
                dw      offset entry
                db      '$TEST$  '      ← 8文字
;
strategy        proc    far
                  :
                  :
                endp
;
entry           proc    far
                  :
                  :
                endp
```

先頭にスペースが入れば大丈夫だろうと思って，“□**TEST**”という名前をつけてしまうことも避けなければいけません．このデバイス名をキーボードから入力することが不可能になってしまいます．

以上のデバイス・ヘッダ項目を満足させればあとは規則的に記述するだけです．**リスト1**にデバイス・ヘッダの例をあげます．ここでは標準入出力デバイスとして“**$TEST$**”という名前のデバイスを定義しています．

デバイス・ドライバ本体

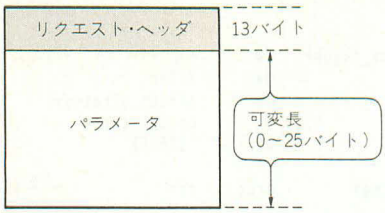
デバイス・ヘッダの属性でスペシャル(SPCL)フラグをonにしなければ，MS-DOSは通常の呼び出し手順でデバイス・ドライバとデータをやりとりします．この一連の過程を，MS-DOSのI/Oリクエスト処理と呼びます．**図8**にI/Oリクエスト処理の概念図を示します．この図をみると，デバイス・ドライバは二つの処理を行わなくてはならないことがわかります．

● **まずストラテジ・エントリが呼び出される**

一つはストラテジ・エントリが呼び出されたときの処理で，ここでデータやデバイス・ドライバへの処理コマンドが渡されます．もう一つは，割り込みエントリが呼び出されたときの処理で，さきほど渡されたデータやコマンドに基づいて実際の処理を行います．

ストラテジ・エントリが呼び出されるとき，データやコマンドはコマンド・パケットという構造で渡されます．パケットとは小包(こづつみ)の意味です．このようなパケットでデータやコマンドをやりとりする方法はよく行われており，MS-NETWORKのようなLAN(Local Area Network)でも採用されています．

〈図9〉コマンド・パケットの構成



〈表2〉リクエスト・ヘッダの内容

1バイト	パケット長(コマンド・パケット全体の長さ)
1バイト	0
1バイト	コマンド・コード(0から16まで)
1ワード	ステータス(I/O処理の結果を返す)
8バイト	DOS予約域

Macintoshのデバイス・ドライバ

Macintoshにも MS-DOS や UNIX のようなデバイス・ドライバが存在します．このデバイス・ドライバは“DRVR”というタイプのリソースをもち，リソース内部で図Aのような構成になっています．MS-DOSユーザにとってリソースという単語は聞きなれないでしょうが，論理セグメントのような概念だと思えばわかりやすいかもしれません．

この4文字の名前のついた論理セグメントが，システムによって管理されているというわけです．Macintoshのデバイス・ドライバは，デバイス・マネージャによって管理されています．

Macのデバイス・ドライバには，大きく分けて，

- ディスク・ドライバ
- サウンド・ドライバ
- シリアル・ドライバ

— ROMに格納

- ハード・ディスク・ドライバ
- SCSIドライバ
- プリンタ・ドライバ
- アップルトーク・ドライバ
- デスク・アクセサリ(DA)

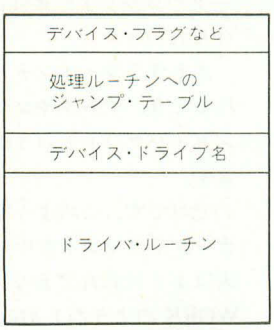
— RAMに格納

があり，これらは標準で装備されています．

ユーザは自由にデバイス・ドライバを作成し，追加することができます．市販のコンパイラ(LightSpeed Cなど)を使うと，デバイス・ドライバのヘッダ部分を自動的に作成してくれますのでたいへん便利です．

Macでユニークなのはデスク・アクセサリ(DA)です．これはアプリケーション実行中に，DAに登録されたほかのプログラムを並列に実行させることができる機能です．

〈図A〉Mac OSのデバイス・ドライバの構造

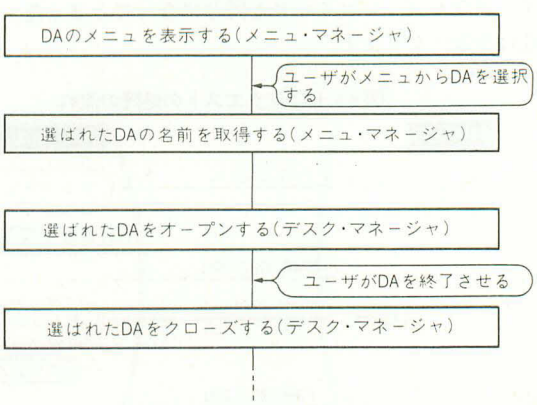


DAは本来のデバイス・ドライバとは動作が異なりますが，デバイス・ドライバとまったく同じ構成をもっています．DAが動作できる環境はアプリケーションが適宜デスク・マネージャを呼び出すことによってつくられます．したがって，デスク・マネージャを呼び出さないアプリケーションの中からDAを利用することはできません(図B)．

これらのデバイス・ドライバに似たもので“INIT”と呼ばれるファイルがあります．MacOSが起動すると最初に“INIT”と名前のついたファイルを実行します．これはちょうどMS-DOSの特殊型デバイス・ドライバの動作に似ています．“INIT”ファイルは実行後メモリに常駐してもかまいません．しかし常駐後にユーザが外部から制御をすることができません．MS-DOSでいえばIOCTL呼び出しができないデバイス・ドライバに相当します．

それに対して外部から制御のできる常駐ソフトには，“CDEV”あるいは“RDEV”というタイプのファイルがあります．“CDEV”あるいは“RDEV”というタイプのプログラムを作るのは“INIT”や“DRVR”を作成するのに比べて多少労力を要します．

〈図B〉DAの操作



● コマンド・パケットで種々のデータとコマンドが渡される

コマンド・パケットは，コマンドの種類やコマンド・パケットの長さなどの情報を入れたリクエスト・ヘッダと呼ばれる固定長部分と，コマンドにより長さが可変のパラメータ受け渡し用の二つの領域から構成されています(**図 9**)．またリクエスト・ヘッダの構成を**表 2** に示します．

MS-DOS からキャラクタ型デバイス・ドライバに渡されるコマンド・コードは 14 種類あり(MS-DOS 3.0 以前では 11 種類)，**表 3** にその一覧を示します．

コマンド・コード 13 と 14 は，デバイス・ドライバの開始処理と終了処理を行わせるコマンド・コードです．起動時以外にデバイス・ドライバを活性化させたり不活性化させたりする場合に使用されるもので，特殊なコマンドです．

たとえば，コマンド・コード 14 の“DEVICE CLOSE”では占有メモリの返却，使用 EMS 領域の解放，取得，割り込みベクタの復帰などのような処理を行い，デバイス・ドライバが動作する以前の状態に資源を戻すようなことを行います．

コマンド・コード 13 の“DEVICE OPEN”は，その反対の処理を行います．コマンド・コード 0 の“INIT”処理と似ていますが，“INIT”は起動時に 1 回しか呼び出されないのに対して，このコードは何回でも呼び出されることがあります．

● I/O コントロールのコードの処理内容は作る人まかせ

コマンド・コード 3 と 12 は，どちらも I/O コントロールに関する処理コマンドです．ほとんどのデバイス・ドライバではサポートする必要はありませんが，なにかデバイス・ドライバに特別な処理を行わせたいとか，内部情報を読みとりたいなどのような場合にのみサポートされます．

MS-DOS では，I/O コントロールによって渡す情報型式や受け取る情報内容はまったく規定されていません．したがって，ユーザは自由にインプリメントすることができますが，逆に未知のデバイス・ドライバが I/O コントロール呼び出しに関してどのような挙動をとるかを判断することはできません．

〈表 3〉コマンド・コードとその内容

0	INIT(デバイス・ドライバの初期設定)
3	IOCTL INPUT(デバイス・ドライバ自身からデータを入力する)
4	INPUT(入力，読み込み)
5	NON-DESTRUCTIVE INPUT NO WAIT(バッファ保存読み込み)
6	INPUT STATUS(入力状況)
7	INPUT FLASH(入力バッファを空にする)
8	OUTPUT(出力，書き込み)
9	OUTPUT(ベリファイ付き書き込み)
10	OUTPUT STATUS(出力状況)
11	OUTPUT FLASH(出力バッファの内容を空にする)
12	IOCTL OUTPUT(デバイス・ドライバ自身へデータを渡す)
13	DEVICE OPEN(デバイスをオープンする)
14	DEVICE CLOSE(デバイスをクローズする)
16	OUTPUT UNTIL BUSY(BUSY になるまでデータを出力する)

＊ 13, 14, 16 は MS-DOS 3.1 以降のみ

〈リスト 2〉内部デバイス・ヘッダのチェーンを作成する

```
vct             struc
_offset         dw      ?
_segment        dw      ?
vct             ends

header1     vct     <0ffffh,0ffffh>         ;このポインタは自分で書き換える
            dw      8000h
            dw      offset strategy1
            dw      offset entry1
            db      '$TEST1$'
;
header2     vct     <0ffffh,0ffffh>         ;このポインタはMS-DOSが決める
            dw      8000h
            dw      offset strategy2
            dw      offset entry2
            db      '$TEST2$'

init        proc
            mov     ax,offset header2       ;header2のオフセットを
            mov     header1._offset,ax      ;header1に格納
            mov     ax,cs                   ;header2のセグメントを
            mov     header1._segment,ax     ;header1に格納
             :
             :
```

● 複数のデバイスへの対応はコマンド・コード 0 (初期化)のときに行う

デバイス・ドライバ・ヘッダの項でふれましたが，複数のデバイスをサポートしたデバイス・ドライバでは，コマンド・コード 0 の初期化が呼び出されたときに，複数デバイスの対応処理を行います．この初期化コマンドが実行された時点で，デバイス・ドライバはメモリ中にローディングされ，アドレスも確定されます．しかし，まだほかのデバイス・ドライバとのデバイス・チェーンはつながれていません．MS-DOS は，**CONFIG.SYS** に記述されたデバイス・ドライバを一つ読み込むごとに初期化コマンドをコールするからです．

そこで，この機会を利用して自分自身がサポートする内部デバイス・ヘッダのチェーンをつなげておきます．具体的には，**リスト 2** のように行います．最後のデバイスのポインタだけが MS-DOS によって確定されます．

〈リスト3〉なにもしないデバイス・ドライバ

```
code            segment
        assume  cs:code, ds:code, es:code, ss:code
;
;
;       >> ストラクチャの定義 <<
vct             struc
_offset         dw      ?
_segment        dw      ?
vct             ends
;
;       >> デバイス・ヘッダ <<
device_header   vct     < 0ffffh,0ffffh >
                dw      8000h
                dw      offset strategy
                dw      offset entry
                db      '$TEST$  '
;
;       >> リクエスト・ヘッダ <<
req_command     dw      offset init             ;init
                dw      offset done             ;media check
                dw      offset done             ;build BPB
                dw      offset done             ;IOCTL input
                dw      offset done             ;input
                dw      offset done             ;non-destructive input
                dw      offset done             ;input status
                dw      offset done             ;input flash
                dw      offset done             ;output
                dw      offset done             ;output
                dw      offset done             ;output status
                dw      offset done             ;output flash
                dw      offset done             ;IOCTL output
                dw      offset done             ;device open
                dw      offset done             ;device close
                dw      offset done             ;removable media
                dw      offset done             ;output until busy
;
data_pointer    vct     <>
;
;***********************************************
;*   デバイス・ドライバ・ストラテジ・ルーチン   *
;***********************************************
;
strategy        proc    far
                mov     cs:data_pointer._offset,bx
                mov     cs:data_pointer._segment,es
                ret
strategy        endp
;
;***********************************************
;*    デバイス・ドライバ・エントリ・ポイント    *
;***********************************************
;
entry           proc    far
                push    ax
                push    bx
                push    cx
                push    dx
                push    si
                push    di
                push    bp
                push    ds
                push    es
                lds     si,cs:data_pointer
                mov     bl,byte ptr 2[si]       ;コマンド・コードを取り出す
                cmp     bl,16                   ;コマンド・コードは正しいか？
                ja      error                   ;    まちがっている
                xor     bh,bh
                shl     bx,1
                mov     ax,offset req_command
                add     bx,ax
                mov     cx,word ptr 18[si]      ;転送文字数を取得
                mov     dx,word ptr 20[si]      ;config.sysの引き数ポインタ
                les     di,dword ptr 14[si]     ;転送アドレスを取得
                mov     ax,cs
                mov     ds,ax
                jmp     [bx]                    ;各処理ルーチンへ
;
;       >> DONEフラグとBUSYフラグをセット <<
busy:           mov     ah,03h
                jmp     short set_flag
;
;       >> DONEフラグとERRフラグをセット <<
error:          mov     al,03h                  ;エラー・コード3をセット
                mov     ah,81h
                jmp     short set_flag
;
;       >> DONEフラグをセット <<
done:           mov     ah,01h
set_flag:       lds     si,cs:data_pointer
                mov     word ptr 3[si],ax       ;ステータス・ワードをセット
                pop     es
                pop     ds
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
entry           endp
;
code_end        label   near                    ;ここまでが常駐する
;
;***********************************************
;*          イニシャライズ・ルーチン           *
;***********************************************
;
init:           mov     ax,cs
                mov     ds,ax
                mov     dx,offset message
                mov     ah,09h
                int     21h
                lds     bx,data_pointer
                mov     byte ptr 13[bx],1       ;ユニット数をセット
                mov     ax,offset code_end
                mov     word ptr 14[bx],ax      ;プログラム最終アドレス
                mov     word ptr 16[bx],cs
                jmp     done
;
message         db      0dh,0ah
                db      "ダミー・デバイス・ドライバが読み込まれました"
                db      0dh,0ah
                db      "$"
;
code            ends
                end
;
```

応用例① 何もしないデバイス・ドライバ

デバイス・ドライバの概要を知るために，まず最初に，何もしないデバイス・ドライバを作成してみることにしましょう．**リスト3**がダミー・デバイス・ドライバのプログラムです．これはデバイス・ドライバのひな型です．

この何もしないデバイス・ドライバは本当に何もしないので，実用にはなりません．たんにユーザ・メモリを食いつぶすだけです．しかし，これを元にして利用すれば，どんなデバイス・ドライバでも容易に記述することができるようになるでしょう．

● INITルーチンはとかげのしっぽ

INITルーチンを気をつけて見てください．ここのルーチンはメモリに常駐しません．初期化が終わったあとには切り捨てられるのです．

なぜなら，MS-DOSは，デバイス・ドライバのINITコマンドを最初の1回しか呼び出さないからです．これによってメモリのむだを多少省くことができます．よく利用されるテクニックですので覚えておいて損はありません．

応用例② 画面に時刻を表示させるデバイス・ドライバ

次に，たんにメモリに常駐するだけの目的のためにデバイス・ドライバの型式をとる例をあげましょう．これは，前に説明した特殊型のデバイス・ドライバに属することになります．

● どうやって起動するか

このようなTSRと変わらない目的をもったデバイス・ドライバはMS-DOSから呼び出されることがないので，なんらかの起動のきっかけを自分自身で探さなければなりません．これを「トリガ」と呼びます．拳銃の引き金の意味です．

特殊型のデバイス・ドライバは，なんらかのイベントによるトリガがなければ何もしません．うまく動作するかどうかは，イベントの選択にかかってきます．

MS-DOSでは，いくつかのイベントが考えられます．

(1) MS-DOSシステム・コールが行われたとき
(2) BIOSコールが行われたとき
(3) なんらかのキーボードが押されたとき
(4) タイマがカウント・アップしたとき
(5) CRTの垂直帰線(VSYNC)のとき
(6) マウス割り込みが生じたとき

ほかにもあるかもしれませんが，これらは代表的なイベントです．

● 割り込みベクタを自分のプログラムに飛ばす……フックするという

これらのイベントが生じたことを感知してデバイス・ドライバを動作させるためには，フックの技法を使います．たとえばMS-DOSのシステム・コールが行われた時点でデバイス・ドライバに制御を渡すには，割り込みベクタ・テーブルの**INT 21H**の部分を本来のMS-DOS内部ではなく，デバイス・ドライバ本体の入口のアドレスへジャンプするように書き直します．デバイス・ドライバは，所定の動作を行った後に何ごともなかったように以前の**INT 21H**のアドレスへジャンプします．

フックする割り込みベクタ・テーブルの書き換えは，デバイス・ドライバの初期化コールの際に行います．

このようなフックの技法では，大きな制約が生じることに注意をしてください．たとえばキーボード押し下げイベントをデバイス・ドライバのトリガに選んだとしましょう．キーボードはいつ押されるかわかりません．たとえば，ディスクをアクセスしている最中かもしれませんし，かな漢字変換の途中ということもありえます．

● トリガは慎重に選ぶ

これらのタイミングを十分認知したうえで，所定の動作を行わなくてはなりません．どのイベントをトリガに使うかは特殊デバイス・ドライバ作成時の最大のポイントであるといえます．

特殊型デバイス・ドライバは，TSRを作成しているときとまったく同じ注意を払って作成しなければならないのです．

このような注意を払って作成したのが，**リスト4**に示す「時刻表示」デバイス・ドライバです．このデバイス・ドライバは**INT 28H**をフックして，その最中に画面最上行に現在時刻を表示します．**INT 28H**という割り込みベクタはMS-DOSの内部で使用されていて，キーボード待ちの状態で繰り返し呼び出される割り込みベクタですので，このような用途には向いているイベントです．

しかし多少のオーバヘッドが発生しています．もっと良いイベントがあるならば，それに変更したほうがよいでしょう．

たとえば，**INT 28H**中に時刻を読みだし，VSYNC(**INT 0AH**)中に表示するなどといったように，一つのイベントですべてを実行させないようにします．こうすることで，割り込みフックによるオーバヘッドを少なくすることができます．

このデバイス・ドライバは，デバイス・ドライバとしては何も機能していないことに注意してください．INITコール以外はすべてエラーを返すようになっています．

応用例③　縮小印字デバイス・ドライバ

標準型のデバイス・ドライバの例として，縮小印字デバイス・ドライバを紹介します．このデバイス・ドライバは“**BPRN**”と呼ばれるデバイス名をもっており，このデバイスへ出力される文字データがプリンタへ縮小印字されます．プリンタはNECのPC-PR201相当を想定しています．これは，縮小印字をグラフィック印字で行うためにプリンタのエスケープ・シーケンスに，PC-PR201のものを使用しているからです．このため，ESC/Pプリンタでは誤動作を起こしますので注意してください．

いままでのデバイス・ドライバの説明をふりかえれば，この縮小印字プリンタがどのように動作するかがわかるでしょう．

なお，あくまでデバイス・ドライバの一例としていますので，エラー・チェックなどを省いてあります．印字不可能な文字が送られてきた場合には，動作がおかしくなる場合があるかもしれません．一般に，十分なエラー・チェック・ルーチンは，本体のプログラムよりも長くなってしまうことがあります．

● 実行ファイルを作る

このプログラムをアセンブルするためには，たとえば，**BPRN.ASM**という名前でプログラムが作成されていたとすると，

```
A>MASM BPRN.ASM ;
A>LINK BPRN ;
A>EXE2BIN BPRN.EXE BPRN.SYS
```

と入力することで，デバイス・ドライバが作成されます．これを**CONFIG.SYS**に登録します．

```
DEVICE=BPRN.SYS
```

という1行を記述し，コンピュータをリセットします．もしくは**ADDDRV.EXE**などを利用して組み込んでもかまいません．以後，

```
A>COPY AUTOEXEC.BAT BPRN
```

とすれば，ファイルの内容がプリンタに縮小印字されます．

PARTNER-S/Hを使ったデバイス・ドライバのデバッグ

通常のMS-DOS用デバッガ(CV，SYMDEBなど)では，デバッガのプログラム自身がMS-DOSのシステム・コール(**INT 21H**など)を使用します．そのため，MS-DOSの内部から呼び出されるデバイス・ドライバ(以下ドライバと略す)の中にブレーク・ポイントを設定すると，ブレーク時にMS-DOSに再入してしまうために，システムが誤動作してしまいます．

一方，PARTNER(京都マイクロコンピュータ製)には，MS-DOSのシステム・コールを一切使用しない状態でデバッグをすることができるインディペンデント・モードがあります．このモードを利用すれば，MS-DOSのシステム内やドライバ内にブレーク・ポイントを設定してシンボリック・デバッグができます．

● ドライバの開発およびデバッグの手順

MS-DOSの規約にしたがってドライバのソフトウェアを記述します．このときに，ドライバの初期化部分でドライバの常駐セグメント(CS)の値を画面に表示するようにしておきます．また，ドライバのアセンブル(コンパイル)の際には，必ずMAPファイルを作ります．

次に，**CONFIG.SYS**にドライバを登録してMS-DOSを再起動するかADDDRVコマンドを使ってMS-DOSに組み込んで登録をします．このとき表示されるドライバの常駐セグメント値をメモします(この段階でうまく立ち上がらないドライバはPARTNERでもデバッグはできない)．組み込みがうまくいくと，**COMMAND.COM**のプロンプト(**A>**など)が表示されます．そこでPARTNERを**−IND**オプションを付けてインディペンデント・モードで起動します．

```
A>PT −IND ↵ ……PARTNERの起動
```

PARTNERの起動後，Lコマンドでドライバの MAPファイルを読み込みます．

```
>L (デバイス・ドライバのMAPファイル),
   (常駐セグメント値) ↵
```

MAPファイルが正常に読み込まれると，シンボルを使ってメモリ参照や逆アセンブル表示およびブレーク・ポイントの設定などができます．

次に，SYSTEMコマンドを実行します．

ここで，PARTNERの常駐が終了し，**COMMAND.COM**の制御に戻ります．ただし，強制ブレークによりいつでもPARTNERに制御を戻すことができます．

ここからドライバの論理デバッグ作業になります．ドライバを使用する操作をして動作チェックをします．そして，ドライバの処理が先ほど設定したブレーク・ポイントに達したところでPARTNERに制御が移ります．このときにメモリ内容，レジスタ値などの参照/変更を行い，**G**(o，**T**(race，**P**(ass，などの実行コマンドを使ってデバッグを進められます．

〈山本彰一〉

〈リスト４〉時刻表示デバイス・ドライバ

```
;
code            segment
                assume  cs:code, ds:code, es:code, ss:code
;
;
;       >> ストラクチャの定義 <<
vct             struc
_offset         dw      ?
_segment        dw      ?
vct             ends
;
header          vct     < 0ffffh,0ffffh >
                dw      8000h
                dw      offset strategy
                dw      offset entry
                db      '$CLOCK!!'
;
data_pointer    vct     < 0ffffh, 0ffffh >      ;ストラテジ・データ
old_int_28h     vct     <>                      ;割り込みトラップ
vram_address    vct     < 0000H, 0a000h >       ;表示領域アドレス
;
year            db      ?                       ;読み出し年
month           db      ?                       ;読み出し月，曜日
date            db      ?                       ;読み出し日
hour            db      ?                       ;読み出し時(0～23 BCD)
minute          db      ?                       ;読み出し分(0～59 BCD)
second          db      ?                       ;読み出し秒(0～59 BCD)
;
;
;*************************************************************************
;*                デバイス・ドライバのストラテジ・ルーチン               *
;*************************************************************************
;
strategy        proc    far
                mov     cs:data_pointer._offset,bx
                mov     cs:data_pointer._segment,es
                ret
strategy        endp
;
;*************************************************************************
;*                デバイス・ドライバのエントリ・ポイント                 *
;*************************************************************************
;
entry           proc    far
                push    ax
                push    bx
                push    cx
                push    dx
                push    si
                push    di
                push    bp
                push    ds
                push    es
                lds     si,cs:data_pointer
                cmp     byte ptr 2[si],0        ;INITコマンドか？
                jnz     entry1                  ;ちがう
                call    init
                mov     word ptr 3[si],0100h    ;正常終了
                jmp     short entry2

entry1:         mov     word ptr 3[si],8103h    ;常にエラーを返す
entry2:         pop     es
                pop     ds
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
entry           endp
;
;*************************************************************************
;*                バック・グラウンド処理割り込みルーチン                 *
;*                          INT 28H                                      *
;*************************************************************************
;
int_28h         proc    near
                push    ax
                push    bx
                push    cx
                push    dx
                push    ds
                push    es
;
;       >> BIOSによる時刻の読み出し <<
                mov     ax,cs
                mov     es,ax
                mov     ds,ax
                mov     bx,offset year
                mov     ah,00h
                int     1ch
;
;       >> 時刻の表示 <<
                mov     ch,hour
                mov     cl,ch
                shr     ch,1
                shr     ch,1
                shr     ch,1
                shr     ch,1
                or      ch,30h
                and     cl,0fh
                or      cl,30h
;
                mov     ah,minute
                mov     al,ah
                shr     ah,1
                shr     ah,1
                shr     ah,1
                shr     ah,1
                or      ah,30h
                and     al,0fh
                or      al,30h
;
                mov     dh,second
                mov     dl,dh
                shr     dh,1
                shr     dh,1
                shr     dh,1
                shr     dh,1
                or      dh,30h
```

〈リスト4〉時刻表示デバイス・ドライバ(つづき)

```
                and     dl,0fh
                or      dl,30h
                lds     bx,dword ptr vram_address
                mov     byte ptr [bx],ch
                mov     byte ptr [bx+2],cl
                mov     byte ptr [bx+4],':'
                mov     byte ptr [bx+6],ah
                mov     byte ptr [bx+8],al
                mov     byte ptr [bx+10],':'
                mov     byte ptr [bx+12],dh
                mov     byte ptr [bx+14],dl
;
                pop     es
                pop     ds
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                jmp     cs:old_int_28h          ;旧INT 28Hへ復帰
int_28h         endp
;
code_end        label   near                    ;ここまでが常駐する
;
;*************************************************************************
;*                          初期化ルーチン                               *
;*************************************************************************
;
init            proc    near
                mov     ax,cs
                mov     ds,ax
                mov     dx,offset message
                mov     ah,09h
                int     21h
;
;       >> 割り込みベクタのトラップ <<
                cli
                xor     dx,dx
                mov     es,dx
                mov     dx,word ptr es:[0028h*4]
                mov     old_int_28h._offset,dx
                mov     dx,word ptr es:[0028h*4+2]
                mov     old_int_28h._segment,dx
                mov     word ptr es:[0028h*4],offset int_28h
                mov     word ptr es:[0028h*4+2],cs
;
                lds     bx,data_pointer
                mov     byte ptr 13[bx],1       ;ユニット数をセット
                mov     ax,offset code_end
                mov     word ptr 14[bx],ax      ;プログラム最終アドレス
                mov     word ptr 16[bx],cs
                mov     word ptr 3[bx],0100h    ;DONEフラグをセット
                ret
init            endp
;
message         db      0dh,0ah
                db      "時刻表示デバイス・ドライバが読み込まれました"
                db      0dh,0ah
                db      "$"
;
code            ends
                end
```

〈リスト5〉縮小印字デバイス・ドライバ

```
;
code            segment
                assume  cs:code, ds:code, es:code, ss:code
;
;       >> ストラクチャの定義 <<
vct             struc
_offset         dw      ?
_segment        dw      ?
vct             ends
;
header          vct     < 0ffffh,0ffffh >
                dw      0c000h
                dw      offset strategy
                dw      offset entry
                db      'BPRN    '
;
;       >> 縮小プリンタのリクエスト・ヘッダ <<
bprn_request    dw      offset init             ;init
                dw      offset done             ;media check
                dw      offset done             ;build BPB
                dw      offset done             ;IOCTL input
                dw      offset done             ;input
                dw      offset busy             ;non-destructive input
                dw      offset done             ;input status
                dw      offset done             ;input flash
                dw      offset bprn_out         ;output
                dw      offset bprn_out         ;output
                dw      offset bprn_status      ;output status
                dw      offset done             ;output flash
                dw      offset done             ;IOCTL output
;
data_pointer    vct     < 0ffffh, 0ffffh >      ;ストラテジ・データ
;
;       >> 漢字プリント制御データ <<
pkanji_first    db      0                       ;漢字第1バイト格納フラグ
pkanji_1st_byte db      0                       ;漢字第1バイト格納エリア
pgdot8          db      1bh,'M',1bh,'I0008$'    ;16ビット・グラフィック・モード
pgdot16         db      1bh,'M',1bh,'I0016$'    ;16ビット・グラフィック・モード
font_buffer     db      20 dup (0)              ;ANKフォント読み出しバッファ
buff            db      2*54 dup (?)            ;文字出力用バッファ
;
;*************************************************************************
;*                デバイス・ドライバのストラテジ・ルーチン                 *
;*************************************************************************
;
strategy        proc    far
                mov     cs:data_pointer._offset,bx
                mov     cs:data_pointer._segment,es
                ret
strategy        endp
;
;*************************************************************************
;*                デバイス・ドライバのエントリ・ポイント                   *
;*************************************************************************
;
entry           proc    far
                push    ax
                push    bx
                push    cx
                push    dx
                push    si
                push    di
```

```
                push    bp
                push    ds
                push    es
                lds     si,cs:data_pointer
                mov     bl,byte ptr 2[si]       ;コマンド・コードを取り出す
                cmp     bl,0ch                  ;コマンド・コードは正しいか
                ja      error                   ;  まちがっている
                xor     bh,bh
                shl     bx,1
                mov     ax,offset bprn_request
                add     bx,ax
                mov     cx,word ptr 18[si]      ;転送文字数を取得
                mov     dx,word ptr 20[si]      ;initルーチンで使用する
                les     di,dword ptr 14[si]     ;転送アドレスを取得
                mov     ax,cs
                mov     ds,ax
                jmp     [bx]                    ;各処理ルーチンへ
;
busy:           mov     ah,03h                  ;DONEとBUSYフラグをセット
                jmp     short set_flag
;
error:          mov     al,03h                  ;エラー・コード3をセット
                mov     ah,81h                  ;DONEとERRフラグをセット
                jmp     short set_flag
;
done:           mov     ah,01h
set_flag:       lds     si,cs:data_pointer
                mov     word ptr 3[si],ax       ;ステータス・ワードをセット
                pop     es
                pop     ds
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
entry           endp
;
;*************************************************************************
;*                  縮小プリンタ・ステータス処理ルーチン                 *
;*************************************************************************
;
bprn_status     proc    near
                mov     ah,12h
                int     1ah
                and     ah,01h
                jz      busy
                jmp     done
bprn_status     endp
;
;*************************************************************************
;*                    縮小プリンタ出力処理ルーチン                       *
;*************************************************************************
;
;入力:  CX = 文字数
;       ES:DI = 出力文字格納アドレス
;
bprn_out        proc    near
                mov     ax,es
                mov     ds,ax
                mov     si,di
                cld
bprn_out1:      lodsb                           ;文字データを得る
                call    bprn_out2               ;コンソール出力ルーチンへ
                loop    bprn_out1
                jmp     done
;
;       >> プリンタ文字出力 <<
bprn_out2:      push    cx
                push    si
                push    ds
                mov     cl,al
                mov     ax,cs
                mov     ds,ax
                cmp     cl,20h                  ;コントロール・コードか
                jb      bprn_out3               ;  そうだ
                call    prn_char                ;プリンタに1文字出力する
                pop     ds
                pop     si
                pop     cx
                ret
;
;       >> コントロール・コード処理ルーチン <<
bprn_out3:      call    line88
                mov     ah,cl
                call    lprint
                pop     ds
                pop     si
                pop     cx
                ret
bprn_out        endp
;
;*************************************************************************
;*                        1/8 インチ改行コード送出                       *
;*************************************************************************
;
line88          proc    near
                mov     ah,1bh
                call    lprint
                mov     ah,'B'
                call    lprint
                ret
line88          endp
;
;*************************************************************************
;*                          文字プリント・ルーチン                       *
;*************************************************************************
;
prn_char        proc    near
                cmp     pkanji_first,1
                jz      prn_char2
;
                cmp     cl,81h
                jb      prn_ank
                cmp     cl,0a0h
                jb      prn_char1
                cmp     cl,0e0h
                jb      prn_ank
                cmp     cl,0fdh
                jae     prn_ank
prn_char1:      mov     pkanji_first,1
                mov     pkanji_lst_byte,cl
                ret
```

```
;
prn_char2:      mov     ch,pkanji_1st_byte
                call    convert_jis
                sub     ch,20h
                call    prn_kanji
                mov     pkanji_first,0
                ret
prn_char        endp
;
;*************************************************************************
;*                      英数字 1 文字プリント・ルーチン                  *
;*************************************************************************
;
;入力： CL = ASCII 文字コード
;
prn_ank         proc    near
                mov     dx,offset pgdot8        ;普通コピー
                call    llist                   ;16ビット・グラフィック・モード
;
;       >> 文字フォントの開始アドレスを求める <<
                call    ank_access
;
;       >> 文字フォントを読みバッファに格納する <<
                mov     cx,16
prn_ank1:       mov     al,byte ptr es:[si]     ;グラフィック・データを得る
                xor     ah,ah
                mov     bx,15                   ;バッファ・ポインタをセット
prn_ank2:       rcr     ax,1                    ;data set to buff
                rcr     byte ptr buff[bx],1     ;
                rcr     byte ptr buff-1[bx],1
                sub     bl,2                    ;decrement buffer pointer
                jns     prn_ank2                ;
;
                inc     si
                loop    prn_ank1
;
;       >> バッファのデータをプリンタに送出する <<
                mov     bx,0                    ;バッファ・ポインタをセット
                mov     cx,8
prn_ank3:       mov     dx,offset buff[bx]      ;load graphic data
                mov     ah,dl
                call    lprint                  ;  & output to printer
                mov     ah,dh
                call    lprint
                add     bx,2
                loop    prn_ank3
                ret
prn_ank         endp
;
;*************************************************************************
;*                      漢字 1 文字プリント・ルーチン                    *
;*************************************************************************
;
;入力： CX = 内部漢字コード (JIS - 2000H)
;
prn_kanji       proc    near
                mov     dx,offset pgdot16       ;普通コピー
                call    llist                   ;16ビット・グラフィック・モード
;
                mov     al,0bh
                out     068h,al                 ;漢字をドット・アクセスにする
                mov     al,cl                   ;漢字文字コードの第2バイト
                out     0a1h,al                 ;漢字キャラジェネのアクセス
                mov     al,ch                   ;漢字文字コードの第1バイト
                out     0a3h,al                 ;漢字キャラジェネのアクセス
                mov     si,00000000b            ;漢字の右側カウンタ
                mov     di,00100000b            ;漢字の左側カウンタ
                cld
;
                mov     cx,16
;
;       >> 漢字のビット・イメージを取得する <<
prn_kanji2:
;
;       >> 漢字の左側を取得する <<
                mov     ax,di
                out     0a5h,al
                in      al,0a9h                 ;パターンを読む
                inc     di
                mov     dh,al
;
;       >> 漢字の右側を取得する <<
                mov     ax,si
                out     0a5h,al
                in      al,0a9h                 ;パターンを読む
                inc     si
                mov     dl,al
;
;       >> 文字フォントをバッファに格納する <<
                mov     ax,dx                   ;グラフィック・データを得る
                mov     bx,31                   ;バッファ・ポインタをセット
prn_kanji3:     rcr     ax,1                    ;data set to buff
                rcr     byte ptr buff[bx],1     ;
                rcr     byte ptr buff-1[bx],1
                sub     bl,2                    ;decrement buffer pointer
                jns     prn_kanji3              ;
                loop    prn_kanji2
;
;       >> バッファのデータをプリンタに送出する <<
                mov     bx,0                    ;バッファ・ポインタをセット
                mov     cx,16
prn_kanji4:     mov     dx,offset buff[bx]      ;load graphic data
                mov     ah,dl
                call    lprint                  ;  & output to printer
                mov     ah,dh
                call    lprint
                add     bx,2
                loop    prn_kanji4
;
                mov     al,0ah                  ;漢字をコード・アクセスにする
                out     068h,al
                ret
prn_kanji       endp
;
;
ank_access      proc    near
                mov     dh,80h
                mov     dl,cl
                mov     cx,offset font_buffer
                mov     bx,cs
                mov     ah,14h
                int     18h
                mov     ax,cs
                mov     es,ax
```

```
                mov     si,offset font_buffer+2
                ret
ank_access      endp
;
kanji_access    proc    near
                mov     al,0bh
                out     068h,al                 ;漢字をドット・アクセスにする
                mov     al,cl                   ;漢字文字コードの第 2 バイト
                out     0a1h,al                 ;漢字キャラジェネのアクセス
                mov     al,ch                   ;漢字文字コードの第 1 バイト
                out     0a3h,al                 ;漢字キャラジェネのアクセス
                mov     si,00000000b            ;漢字の右側カウンタ
                mov     di,00100000b            ;漢字の左側カウンタ
                cld
                ret
kanji_access    endp
;
;       >> 文字列のプリンタ出力 <<
llist           proc    near
                push    bx
                mov     bx,dx
llist1:         mov     ah,[bx]
                cmp     ah,'$'
                jz      llist2
                call    lprint
                inc     bx
                jmp     llist1
llist2:         pop     bx
                ret
llist           endp
;
;       >> 1バイト・データのプリンタ出力 <<
lprint          proc    near
                push    ax
                mov     al,ah
                mov     ah,11h
                int     1ah
                pop     ax
                ret
lprint          endp
;
;*****************************************************************
;*                  シフト JIS から JIS への変換                 *
;*****************************************************************
;
;入力: CX = シフト JIS コード
;出力: CX = JIS C6226コード
;
convert_jis     proc    near
                cmp     ch,80h                  ;if(ch == 0x80)
                jz      convert_jis6            ;       return;
                cmp     ch,0a0h                 ;else if(ch < 0xa0)
                jae     convert_jis1            ;       ch -= 0x70;
                sub     ch,70h                  ;
                jmp     short convert_jis2      ;
                                                ;else
convert_jis1:   sub     ch,0b0h                 ;       ch -= 0xb0;
convert_jis2:   or      cl,cl                   ;if(cl <= 0x80)
                jns     convert_jis3            ;       --cl;
                dec     cl                      ;
convert_jis3:   add     ch,ch                   ;ch *= 2;
                cmp     cl,9eh                  ;if(cl >= 0x9e)
                jb      convert_jis4            ;       cl -= 0x5e;
                sub     cl,5eh                  ;
                jmp     short convert_jis5      ;
                                                ;else
convert_jis4:   dec     ch                      ;       --ch;
convert_jis5:   sub     cl,1fh                  ;cl -= 0x1f;
convert_jis6:   ret                             ;
convert_jis     endp
;
;
code_end        label   near                    ;ここまでが常駐する
;
;*****************************************************************
;*                    イニシャライズ・ルーチン                   *
;*****************************************************************
;
init:           mov     ax,cs
                mov     ds,ax
                mov     dx,offset message
                mov     ah,09h
                int     21h
                lds     bx,data_pointer
                mov     byte ptr 13[bx],1       ;ユニット数をセット
                mov     ax,offset code_end
                mov     word ptr 14[bx],ax      ;プログラム最終アドレス
                mov     word ptr 16[bx],cs
                jmp     done
;
message         db      0dh,0ah
                db      "縮小印字デバイス・ドライバが読み込まれました"
                db      0dh,0ah
                db      "$"
;
code            ends
                end
```

第4章 EMMマネージャを使うための基本をマスタする

拡張メモリ用EMSドライバの使い方

MS-DOS 3.3からEMSDRVER(PC9801)という拡張メモリ用デバイス・ドライバが付属しています．メモリ・ボード・メーカにもユーティリティ・ソフトとしてこのEMSドライバがついてきます．ここでは，実例を示しながら，そのドライバを使うためのプログラミングを説明します．

菱 博嘉

最近，いろいろな雑誌のメーカ広告などで，何かと話題のEMS(Expanded Memory Specification)です．EMSについて“メモリの一種である”とか“アプリケーションで，EMSを使うものがある”といったことは知られているようです．しかし，実際にプログラムなどに応用しようとすると，これといった具体例が少ないこともあり，よく使い方がわからないといったことも多いのではないでしょうか？

ここでは，EMSに関し，大体の知識はあるのだが，実際に使うにはどうしたらよいのだろう??といった疑問をお持ちの方を対象にして，プログラムの実例をあげながら，一般的なツール(例えばC言語)のみで，EMSメモリをどのように使っていけばよいのかといった点を中心に紹介します．

なおEMSは，1984年にLotus, Intel, Microsoftの3社によって発表されたので，LIM EMSとも呼ばれます．

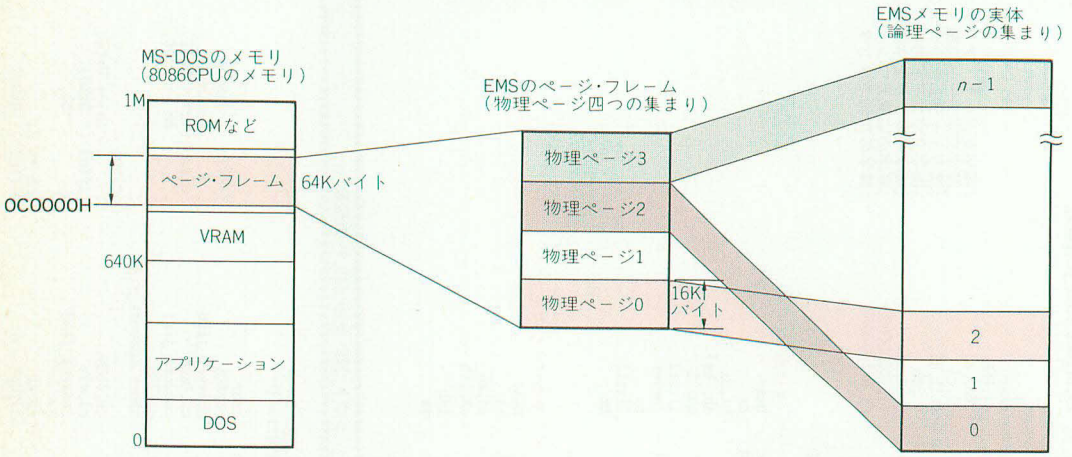
EMSとは何か?

EMSとは，一言でいって規格のことです．ちょうどEMSは，OSのファンクション・コールの利用を定めた規格であり，その対象が主に拡張メモリとなっていると考えるとよいでしょう．

従って，EMSは，アプリケーションと拡張メモリのインターフェース部分を定めた文章ということになります．EMSにもOSと同じようにバージョンがありますが，ここでは，日本で一般的なEMS V 4.0を基に進めることにします．

EMSメモリというのは，このEMSの規格に従って，アプリケーションからアクセスするのに適したハードウェアです(図1)．

〈図1〉MS-DOSのメモリとEMSメモリの関係



★プログラムから見るEMSのメモリは，ページ・フレームを窓として物理ページへ現れる．
ページ・フレームの位置(アドレス)は，EMMへのファンクション・コールにより知る．PC9801では通常0C000Hセグメント．

★ページ・フレームは四つの独立した物理ページから成り立つ．
一つの物理ページはCPUメモリ空間の16Kバイトをしめる．
従ってページ・フレームは，16Kバイト×4＝64Kバイトとなり，ちょうど8086CPUの1セグメント分となる．

★物理ページ(CPUのアクセスするEMSメモリの窓)と，論理ページ(16Kバイト単位で管理されるEMSメモリ)は，相互に独立した関係をもつ．従って，同一の論理ページは，同時に複数の物理ページを窓として，アクセスすることができる．

★EMSメモリは16Kバイト単位で論理ページという単位で区切られる．
従って，1MバイトのEMSメモリは，
16Kバイト×64＝1024K
(1Mバイト)
となり，64個の論理ページとなる(0～63)．

● EMM デバイス・ドライバを組み込む

一般的に，ハードウェアとアプリケーション間で何等かのインターフェースが決まると，それを実現するためのOS/ソフトウェアが必要となってきます．この，OSに相当するソフトウェアが，EMSではEMM(拡張メモリ・マネージャ，以下EMM)です．

EMMは，多くの場合デバイス・ドライバとしてOS(MS-DOS)の中に組み込まれ，アプリケーションはEMSメモリへアクセスする場合，EMSの規格に従ってEMMへ必要に応じてリクエストを出す形をとります．

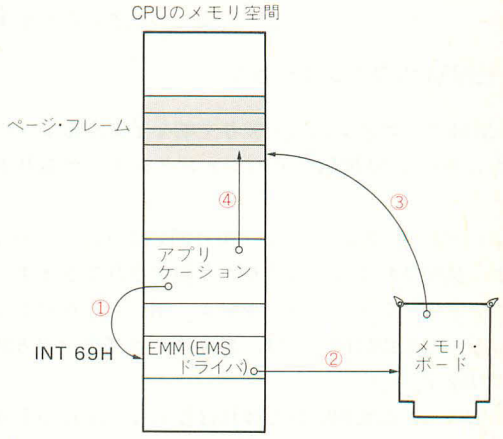
実際，個々のEMMファンクション・リクエストを見ると，まさにOSのファイル関連のリクエストと，メモリ管理関連のリクエストを合わせたような構成となっていることに気がつきます．EMSファンクションの一部(今回の例題で使用した部分)を**表1**に示します．

◆ EMMリクエスト/EMSメモリへのアクセス

アプリケーションからEMMへのリクエスト・コール(以下EMMリクエスト)は，MS-DOSのCPUである8086のソフトウェア割り込み**67H**を用いて行います．すべてのEMMリクエストは，この割り込み(**INT 67H**)をとおして行います(**図2**)．

EMMのファンクション全体を見てみると，EMMリクエストそのものが，EMSメモリそのものへのアクセ

〈図2〉アプリケーションとEMM/EMSメモリ・アクセスの関係(インターフェース)



① アプリケーションは，EMMを**INT 67H**を通して呼び出す．
② EMMは，アプリケーションの要求に従って，メモリ・ボードを操作し，
③ メモリ・ボード上のメモリを，CPUメモリ空間の窓(物理ページ)へ出す．
④ アプリケーションは，物理ページへアクセスを行うことによって，EMSメモリへアクセスする．

〈表1〉例題で使用するEMSファンクション

ファンクション番号	内　容	入　出　力　条　件
1	EMMのステータスの取得	IN: AH＝**40H** OUT: AH＝**0** 正常終了
2	ページ・フレーム・セグメントの取得	IN: AH＝**41H** OUT: AH＝**0** 正常終了 BX＝ページ・フレームのセグメント値
3	未使用ページ数の取得	IN: AH＝**42H** OUT: AH＝**0** 正常終了 BX＝未使用の(これから使用できる)EMSページ数 DX＝EMSメモリ全体のページ数
4	EMSページ割り当てとハンドルの取得	IN: AH＝**43H** BX＝割り当てたいERSメモリのページ数 OUT: AH＝**0** 正常終了 DX＝以降のEMMリクエストで使用するEMMハンドル(数値)
5	EMSメモリのマップ(CPUメモリ空間に現す)． 同時に，現在現れているメモリをCPUメモリ空間から消す	IN: AH＝**44H** AL＝論理ページを割り当てるCPUメモリ空間の物理ページ番号 BX＝ALで指定した物理ページに割り当てる(CPUメモリ空間に現す)EMSメモリの論理ページ番号 DX＝ファンクション4で所得したEMMハンドル OUT: AH＝0 正常終了
6	EMSページの解放とハンドルの返却	IN: AH＝**45H** DX＝ファンクション4で所得したEMMハンドル OUT: AH＝**0** 正常終了
7	EMMのサポートするEMSのバージョンの取得(今回は未使用)	IN: AH＝**46H** OUT: AL＝バージョン番号(BCDで，4.0であれば**40H**となる)
8	EMSの現在のマップ状態を保存(状態のPUSH)	IN: AH＝**47H** DX＝ファンクション4で所得したEMMハンドル OUT: AH＝**0** 正常終了
9	EMSのマップ状態の復元(ファンクション8でPUSHした状態のPOP)	IN: AH＝**48H** DX＝ファンクション4で所得したEMMハンドル OUT: AH＝**0** 正常終了

スを伴うものが極めて少ないことに気がつきます．これは，EMSで扱うメモリというデバイス自体が，CPUから直接アクセスでき，かつ高速で，そのため，アクセスごとにたとえ1バイトであってもEMMリクエストをとおして操作するよりは，メモリの実体へのアクセスは，アプリケーションから直接行ったほうが，比較にならないくらい高速となるためと考えられます．

◆ EMMのファンクション

EMMリクエストには，参考文献などにみるように，大小合わせて50種類以上のファンクションがあります．

OSのファンクション・コールと同じように，このうちよく使用されるものは，かなり限定されてきます．どのファンクション・コールがよく用いられるのかといったような点に関しては，多分にプログラマの方針によります．

るにあたって最低限行わなければならない項目があり，これはEMS自体に定められているものです(**表2**)．

● EMSべからず集

EMSの中には，インターフェース仕様と最低限の必要事項，と同時に行ってはいけないこと，いわゆる“べからず”といったような内容も記載されています．

これらは，アプリケーションからEMSメモリを使用する場合，最低限守らなければならないことであり，これに違反しているアプリケーションは，実動作で何が起きても不思議はありません(**表3**)．

プログラム例

例として示すプログラムは，C言語で記述した“ファイルを見る”プログラムです．ちょうど，DOSのTYPEコマンドを，スクリーン・エディタのように，キー操作によってファイルの前後を自由に見たり，表示位置をキー操作によって1行前/後ろへ移動する簡単なプログラムです．

要するに参照するだけのスクリーン・エディタですが，ここでは，データ・バッファをEMSメモリ内にとることによって，EMSメモリの許す限りの大きさのデータを，一度にメモリ上で処理できます．

◆ プログラム方針－1

今回のプログラムは例題ですので，特定の処理系に依存する内容は，極力少なくなるように考えます．

高級言語からEMSメモリを使用したりアクセスする場合，ポイントとなるのは，

(1) EMMリクエストをどのように処理するか
(2) EMSメモリへのアクセスをどう行うか

の2点に絞られてきます．(1)のEMMリクエストに関しては，

▶**INT 67H**を使用すること
▶セグメント・レジスタを含むポインタによってパラメータの受け渡しを行う場合があること
▶ソフトウェア割り込みを扱う関数が必ずしも一般的ではない

などから，必要なEMMリクエストに対応する関数は，アセンブラで記述します．

(2)のEMSメモリへのアクセスに関しては，メモリ・モデルをいわゆるラージ・データとしてもつモデルの利用を考えます．高級言語のポインタにEMSメモリのページ・フレーム・セグメントを直接代入して，このポインタを通したアクセスによって，EMSメモリを高級言語から直接アクセスします(**図3**)．

◆ プログラム方針－2

次に，どのEMMリクエストを用いるかという問題

〈表2〉(1) EMSアプリケーション必須事項(EMS4.0規格書より)

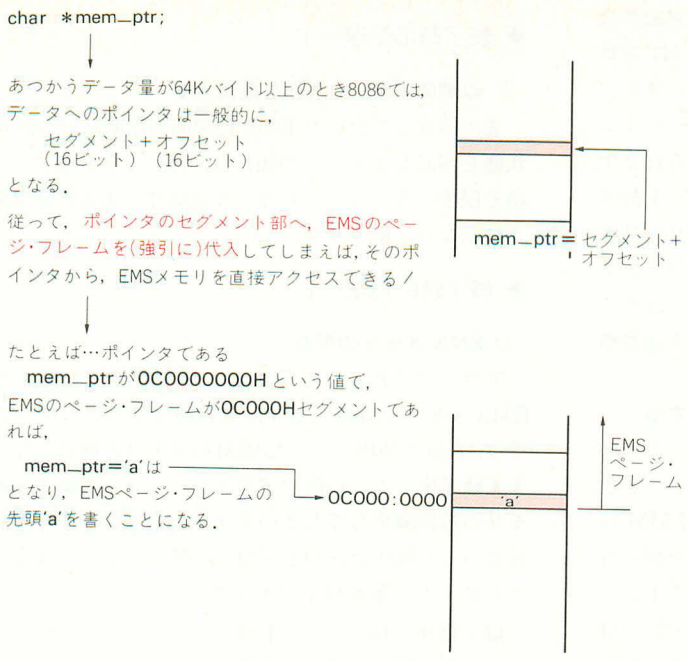
EMSメモリを使用するアプリケーションは，次に記された内容を，その順番に実行しなければならない．

(1) EMMが常駐しているかどうか調べる．
(2) アプリケーションの動作に十分なEMSメモリがあるかどうか調べる(ファンクション3)．
(3) EMSメモリを割り当てる(ファンクション4または18)．
(4) ページ・フレームのベース・アドレスを調べる(ファンクション2)．
(5) EMSのページをページ・フレームにマップする(ファンクション5または17)．
(6) ページ・フレームにマップされたEMSメモリに対し，読み/書き/プログラム実行などを普通のメモリと同じように行う．
(7) 終了前に，割り当てられていたEMSメモリを解放する(ファンクション6または18)．

〈表3〉(1) EMSべからず集(EMS4.0規格書より)

(1) EMSメモリ内にスタックを置いてはならない．
(2) **Int 67H**を書き換えてはならない．
(3) アプリケーションが自分のものではない640Kバイト以下のメモリへEMSメモリをマップしてはならない．
(4) 複数の物理ページで同一EMSメモリをマップし，data aliastingを用いる場合には，EMSハードウェアかエミュレーションであるかを調べなければならない．
(5) アプリケーションは，終了時に必ず自分の使用していたEMSメモリをEMMに返さなければならない．
(6) 常駐プログラム(デバイス・ドライバ)は，EMSのマップ状態を変更する前に，必ず変更前の状態を保存し，変更後には復元しなければならない．
(7) パラメータとして，データ・ストラクチャへのポインタを使用するファンクションでは，データ・ストラクチャのメモリは，マップ・アウト(EMSのマッピングによってCPUメモリから外れること)される可能性のあるメモリに置いてはならない．

〈図3〉EMS メモリへのアクセス方法



ですが，今回は，EMS メモリをデータ・メモリとして用いる点，そしてなるべく簡単な例とするために，**表1**に示したファンクションを用います．

国内の EMS の主流は 4.0 となっていますが，今回の例のような内容であれば，とくに 4.0 のファンクションを用いる必要はなく，EMS 3.2 の単純なファンクションのみでも十分です．

また，実際のアプリケーションでも，実用上，EMS 4.0 のファンクションを使う必要が絶対にあるというものでもありません．むしろ，EMS 3.2 のファンクションのみを使うほうが，プログラム自体は単純に作ることができるでしょう．EMS 4.0 独自のファンクションをを用いた例に関しては，別の機会に紹介したいと考えます．

◆ その他の方針

例題のプログラムでは，EMS 関連以外の処理はすべて標準的な手続きのみで行っています．また，表示時の画面処理はほとんど行っていませんので，1行が80桁を越えるテキストなどでは，画面が見苦しくなる場合もあります．使用する処理系は，PC9801 で一般的な Turbo-C と MASM を使用しました．

このプログラムで用いた手法は，EMS メモリと EMM がインストールされた環境であれば，すべての MS-DOS コンピュータで応用が可能です．

もっとも，PC9801 の Turbo-C で作成したプログラムは，J3100 などのコンピュータでは動作しませんでした．

プログラムの説明

次に，**リスト1** (p.113)の説明をしていきます．

以下のプログラム説明で，各項目に付いた番号と文章中の○囲みの数字は，リストのコメントの数字と対応しています．リストと合わせて参照してください．

① EMS/EMM の有無を調べる

アプリケーションから EMS を使用する場合，まず必要となるのが，EMM がメモリ中に存在するかどうかを知ることです．これは，EMM インターフェースが **INT 67H** を用いているためであり，EMM が存在しない環境で **INT 67H** を発行することは，システムの正常動作を考えるうえで，相当に危険な動作であるからです．

EMM の存在を確認する方法としては，例題中(**リスト1**中“**_ems_inst_chk**”)で用いた EMM に対するデバイス・オープンに依るものが一般的といえるでしょう．

例題以外にも，**INT 67H** のセグメントから，EMM のロードされたセグメントを調べ，EMM のデバイス名を文字列サーチする方法もありますが，プログラム自身の管理外のメモリを直接参照することとなり，余りお勧めできる方法とはいえないでしょう．

◆ EMM の環境を調べる

EMM の存在が確認できたら，次に，

▶EMS メモリのアドレス

▶使用可能なメモリの量

などの EMM のサポートしている EMS のバージョンなど，EMM によって提供される環境を調べます．

② EMS メモリのアドレス

EMS メモリのアドレスは，実際に EMS メモリへアクセスする際に使用するポインタとして用います．

③ 使用可能なメモリ量

使用可能なメモリ量は，アプリケーションが EMS メモリをどれだけ使用可能であるかを示します．例題では，512 K バイトをメモリの下限としています．

EMS のバージョンのチェックは，今現在でもすべての EMS 環境が EMS 4.0 とはいえない(ユーザが，古い EMS エミュレータを使用している可能性もある)ことから必要となります．EMS のバージョンが 4.0 ではなかった場合には，4.0 特有のファンクション

は使用できません．バージョンが4.0ではなかったような場合は，必要なEMMリクエストが実行可能ではないので，EMSメモリを使用しないかもしくは，プログラム自体の実行を中断するなどの処理を行います．

実際のアプリケーションでは，EMSのバージョンが4.0でなければならない理由はないと考えられます．しかし，EMMハンドルのみを取得して，メモリを割り当てずにしておくとか，一度割り当てたメモリの量を後から調節するなどでは，どうしても4.0のファンクションを使用することになります．今回の例ではとくに必要がなかったので，バージョン・チェックは省略しました．

④ EMSメモリを割り当て/ハンドルを取得する

以上の前準備が終了したら，実際に使用するためにEMSメモリの割り当ての要求を行います．このとき，同時に以降のEMMリクエストで使用するEMMハンドルを取得します．EMS 4.0では，メモリを割り当てずに，ハンドルのみを取得することも可能ですが，例題では，あらかじめ割り当て量を決めておいて，1Mバイト(64ページ×16Kバイト)を割り当てています．

例題のような実行終了型のプログラムの場合，このような操作はとくに必要ありませんが，常駐型のプログラムの場合には，必要に応じて使用するEMSメモリの量を調節することがあるでしょう．

⑤ 現在のEMSの状態を保存する

EMSメモリをCPUメモリ空間に配置して，実際に使用する前に，現在のEMSに状態を保存しておきます．これは，例題のような実行型アプリケーションではとくに必要ないのですが，子プロセスとして起動されたような場合を考えて，元の状態に戻せるよう準備しておきます．

⑥ EMSメモリへのアクセス

例題では，実際のEMSメモリへのアクセスでは，ポインタを使用しています．このポインタは，メモリの実体を含むEMSの論理ページとメモリの実体が，実メモリ・アドレス空間で現れるアドレスを要素とする構造体となっています．実際の操作に関しては，**リスト1**と**図3**を参照してください．

◆ 終了時の処理—1

⑦ 以前のEMSの状態を復元する

先に保存しておいたEMS使用前の状態に，EMSの状態を復元します．この操作は，ほかのプロセスの環境を侵害しないといったような意味で，先の状態保存と対になって働きます．

◆ 終了時の処理—2

⑧ EMSメモリの解放

プログラムが終了する前に，自分が使用していたEMSメモリを解放します．実行終了型のプログラムでは，自分の使用していた(EMS)メモリを解放しないまま終了すると，以降のアプリケーションが使えるメモリの量が減少してしまいます．取得したメモリの解放といった操作は，何もEMSに限ったことではなく，プログラムの基本的なマナーです．

以上簡単な例を元に，EMSメモリのアプリケーションからの使用方法を見てきました．プログラム自体はあまり実用的とはいえませんが，アプリケーションなどからEMSを使用する場合の参考にでもなればさいわいです．

また，今後は各種言語処理系も，それ自体EMSに対応したものが発表されてくることと思います．最大32Mバイトという EMSメモリは，今後のOS/2などを待つまでもなく，アプリケーションの扱えるデータ量の制限を事実上なくしてしまうものであるといってもいいすぎではないでしょう．

◆ 参考文献 ◆

(1) LIM EMSの規格書 Ver 4.0 300275-005, LOTUS/INTEL/MICROSOFT EXPANDED MEMORY SPACIFICATIN, Oct. 1987.

(2) レイ・ダンカン；8086マシンのメモリを8Mバイト拡張するEMS，日経バイト，1987年3月．

(3) MS-DOS環境下で32Mバイトの拡張メモリをサポートするEMS 4.0，日経バイト，1988年5月．

機種依存性などに関して

本編の主題とは直接関係がないのですが，PC9801上のTurbo Cで作成したプログラムを，筆者のJ3100SS (DynaBook　メモリ3.5Mバイト) で動かそうとしたところ，動作しませんでした．原因は，Turbo Cの初期化ルーチン，またはライブラリに，機種依存性があるためと考えられます．実際筆者が日常的に使用しているMETAWARE社のHi-Cで作成したプログラムは，DynaBookでもPC9801でも正常に動作します．

機種依存性を考える場合には，どのようなプログラムを組むかだけではなく，使用する処理系の特性も考慮する必要があるようです．

EMSが，機種依存性が低いものであるだけに，ある処理系が，処理系そのものの実行環境のみならず，それによって作成したプログラムまで，処理系の実行環境の影響を受けるというのは残念な気がします．

〈リスト１〉ファイルをみるプログラム(アセンブラ部)

```
        page    60,132
;
;
;       emm handler for test program
;       Aug. 5 1989    :  by doshir.
;       Aug. 8 1989  Turbo-C V2.0 interface
;
;       注意点：
;   1   Turbo-Cでは，プロトタイプ宣言した関数名（アセンブラなど）に
;       無条件で '_' を付けて参照する．従って，ここでは全ての関数名
;       に '_' を付けている．
;   2   アセンブルするときは，関数名(publicラベル)が，大文字変換され
;       ないように，'/MX' スイッチを付けてアセンブルする必要がある．
;
DOS     equ     21h
EMM     equ     67h
;
_TEXT   segment byte public 'CODE'
assume  cs:_TEXT
;
;       ①  ＥＭＳ／ＥＭＭの有無を調べる  （デバイス・オープンによる）
;
emm_name        db      'EMMXXXX0',0
;
        public  _ems_inst_chk
_ems_inst_chk   proc    near
        push    ds
                mov     ax,cs
                mov     ds,ax
                lea     dx,cs:emm_name
                mov     ax,3d00h
                int     DOS             ; ＥＭＭをデバイス・オープンする
                mov     bx,ax
                jc      inst_chk_fault
                mov     ax,4400h
                int     DOS             ; オープンしたものに関する情報の取得
                jc      inst_chk_fault
                mov     ah,3eh
                int     DOS
                jc      inst_chk_fault
                test    dx,0080h        ; デバイスであるかどうかのチェック
                mov     ax,-1
                jnz     inst_chk_end
        inst_chk_fault:
                mov     ax,0
        inst_chk_end:
        pop     ds
        ret
_ems_inst_chk   endp
;
;       ②  ＥＭＳのページ・フレームを取得する
;
        public  _ems_get_page_frame
_ems_get_page_frame     proc    near
        mov     ah,41h
        int     EMM
        or      ah,ah           ; ＯＫであれば，ＢＸにページ・フレームの
        jz      get_page_ok     ; セグメント値が帰ってくる．
         mov    bx,0
get_page_ok:
        mov     dx,bx   ; Ｌｏｎｇのポインタとして Ｃのプログラムへ返す．
        mov     ax,0
        ret
_ems_get_page_frame     endp
;
;       ③  アプリケーションで使用可能なＥＭＳメモリの量を調べる．
;           ここでは，実際に未使用のＥＭＳページ数を知る．
;
        public  _ems_get_free
_ems_get_free   proc    near
        mov     ah,42h
        int     EMM
        or      ah,ah           ; ＢＸ＝未使用ページ数
        jz      get_free_ok     ; ＤＸ＝全ページ数（こちらは使わない）
         mov    bx,0
get_free_ok:
        mov     ax,bx
        ret
_ems_get_free   endp
;
;       ④  ＥＭＳメモリを割り当て／ハンドルを取得する
;
        public  _ems_allocate
_ems_allocate   proc    near
        push    bp
        mov     bp,sp
        mov     ah,43h
        mov     bx,[bp + 4]     ; 引数の必要ページ数
        int     EMM
        or      ah,ah
        jz      alloc_ok
         mov    dx,0
alloc_ok:
        mov     ax,dx           ; ＥＭＭハンドルを返す
        pop     bp
        ret
_ems_allocate   endp
;
;       ⑥  ＥＭＳメモリ（論理ページ）をＣＰＵメモリ空間（物理ページ）に
;           割り当てる（マップする）
;
        public  _ems_map
_ems_map        proc    near
        push    bp
        mov     bp,sp
        mov     ax,[bp + 4]     ; 物理ページ番号
        mov     ah,44h
        mov     bx,[bp + 6]     ; 論理ページ番号
        mov     dx,[bp + 8]     ; ＥＭＭハンドル
        int     EMM
        or      ah,ah
        jz      map_ok
        mov     dx,0
map_ok:
        mov     ax,dx
        pop     bp
        ret
```

〈リスト1〉ファイルをみるプログラム(アセンブラ部,つづき)

```
_ems_map        endp
;
;       ⑧ ＥＭＳメモリの解放
;
        public  _ems_deallocate
_ems_deallocate proc    near
        push    bp
        mov     bp,sp
        mov     dx,[bp + 4]     ; ＥＭＭハンドル
        mov     ah,45h          ; このハンドルによるＥＭＭアクセスの
        int     EMM             ; 終了を宣言する
        or      ah,ah
        jz      dealloc_ok
        mov     dx,0
dealloc_ok:
        mov     ax,dx
        pop     bp
        ret
_ems_deallocate endp
;
;       ⑤ 現在のＥＭＳの状態をセーブする
;
        public  _ems_save_pages
_ems_save_pages proc    near
        push    bp
        mov     bp,sp
        mov     dx,[bp + 4]     ; ＥＭＭハンドル
        mov     ah,47h
        int     EMM
        or      ah,ah
        jz      save_ok
        mov     dx,0
save_ok:
        mov     ax,dx
        pop     bp
        ret
_ems_save_pages endp
;
;       ⑦ 以前のＥＭＳの状態を復元する
;
        public  _ems_restore_pages
_ems_restore_pages      proc    near
        push    bp
        mov     bp,sp
        mov     dx,[bp + 4]     ; ＥＭＭハンドル
        mov     ah,48h
        int     EMM
        or      ah,ah
        jz      restor_ok
        mov     dx,0
restor_ok:
        mov     ax,dx
        pop     bp
        ret
_ems_restore_pages      endp
;
;
_TEXT   ends
        end
```

〈リスト2〉ファイルをみるプログラム(C部)

```
/*
    tl.c
    Apr. 6,15  1988 :   doshir.
    Aug. 5     1989 :   Turbo - C imprement.
 */


#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>


#define EMS_REQPAGES    64  /* EMS pages this program uses */
#define COLMNLIMIT 128      /* 128 column limit */
#define LINES_EMSPAGE 16 * 1024 / COLMNLIMIT    /* lines per 1 ems page */
#define MAXLINES    LINES_EMSPAGE * EMS_REQPAGES
                                /* 128 column limit for 1MB memory */

#define LINES_SCREEN    23  /* lines per 1 screen display */
#define LINES_SKIP  20      /* page next/prev */

char    *map_base;      /* ems page frame base addr */
int free_pages;     /* ems free (usable) pages */
int emm_handle;     /* emm handle number */

/*
    external assembler written routines.
 */

extern  int ems_inst_chk();
extern  char*   ems_get_page_frame();
extern  int ems_get_free();
extern  int ems_allocate(int p);
extern  int ems_map(int p,int l, int h);
extern  int ems_deallocate(int h);
extern  int ems_save_pages(int h);
extern  int ems_restore_pages(int h);

/*
   structure define / for EMS memory access
 */

typedef struct {
    int ems_page;
    char* mem_ptr;
} _line_ptr;

_line_ptr _str_ptr[MAXLINES];   /* is this enough ??? */

void    main (argc, argv)
int argc;
char    *argv[];
{
    int line_no, _page;
    int abort;
```

〈リスト2〉ファイルをみるプログラム(C部,つづき)

```
int cur_line, cur_page;
int disp_line;
char    _ch;
int line_limit;
char    *wrk_ptr;
FILE    *fp;

if (argc < 2) {
    printf ("¥nファイル名が指定されていません.¥n");
} else if ((fp = fopen (argv[1], "r")) == NULL) {
    printf ("¥nファイル %s がオープンできません¥n",argv[1]);
} else if (!ems_inst_chk()) {                          /* ① */
    printf ("¥nＥＭＭドライバがインストールされていません.¥n");
} else if ((map_base = ems_get_page_frame()) == NULL) {    /* ② */
    printf ("¥nＥＭＳページフレームの取得でエラーが生じました.¥n");
} else if ((free_pages = ems_get_free()) == 0) {       /* ③ */
    printf ("¥nＥＭＳの未使用ページが有りません.¥n");
} else if (free_pages < EMS_REQPAGES) {
    printf ("¥nＥＭＳのページが不足です.¥n");
} else if ((emm_handle = ems_allocate(EMS_REQPAGES)) == 0) { /* ④ */
    printf ("¥nＥＭＭハンドルの取得／アロケートでエラーが生じました.¥n");
    exit (-1);
} else if (!ems_save_pages(emm_handle)) {              /* ⑤ */
    printf ("¥nＥＭＳマップ状態のセーブでエラーが生じました.¥n");
    } else {
        abort = 0;
        for (wrk_ptr = map_base, _page = line_no = 0, ems_map(0,_page,emm_handle) /* ⑥ */
            ;line_no < MAXLINES; line_no++ ) {
            _str_ptr[line_no].ems_page = _page;
            _str_ptr[line_no].mem_ptr = wrk_ptr;
            if (fgets (_str_ptr[line_no].mem_ptr, COLMNLIMIT, fp) == NULL)
                break;
            if ((line_no & 0x3f) == 0) {
                wrk_ptr = map_base;
                _page ++;
                if (!ems_map (0, _page, emm_handle)) {     /* ⑥ */
                    printf ("¥nＥＭＳメモリのマップでエラーが生じました.¥n");
                    abort = -1;
                    break;
                }
            } else {
                wrk_ptr += COLMNLIMIT;
            }
        }
        abort = -1;
        line_limit = line_no;
        cur_page = -1;
        _page = 0;
        cur_line = line_no = 0;
        abort = 0;
        while (!abort) {
        printf ("¥xlb[0;0H");
        for (line_no = cur_line, disp_line = 0;
            disp_line < LINES_SCREEN;
            line_no ++, disp_line++) {
            if (line_no < line_limit) {
                if (cur_page != _str_ptr[line_no].ems_page) {
                    cur_page = _str_ptr[line_no].ems_page;
                    if (!ems_map (0, cur_page, emm_handle)) { /* ⑥ */
                        printf ("¥nＥＭＳメモリのマップでエラーが生じました.¥n");
                        abort = -1;
                        break;
                    }
                }
                printf ("¥xlb[K¥r%s", _str_ptr[line_no].mem_ptr);
            } else {
                printf ("¥xlb[K¥n");
            }
        }
        if (!abort ) {
            printf ("+ = next page, - = prev page, ESC = exit");
            _ch = getch();
            switch (_ch) {
                case 0x1b:  abort = -1;
                            break;
                case 0x0d:
                case ' ' :  if ((cur_line + LINES_SCREEN) < line_limit)
                                cur_line ++;
                            break;
                case 0x08:  if (cur_line)
                                cur_line --;
                            break;
                case '+':   if ((cur_line + LINES_SKIP) < line_limit)
                                cur_line += LINES_SKIP;
                            else if (line_limit <= LINES_SCREEN)
                                cur_line = 0;
                            break;
                case '-':   cur_line -= LINES_SKIP;
                            if (cur_line < 0)
                                cur_line = 0;
                            break;
            }
        }
    }
    if (!ems_restore_pages(emm_handle))                /* ⑦ */
    printf ("¥nＥＭＳマップ状態のリストアでエラーが生じました.¥n");
}
if ((emm_handle) && (!ems_deallocate(emm_handle)))     /* ⑧ */
    printf ("¥nＥＭＭハンドルの解放／デアロケートでエラーが生じました.¥n");
if  (fp != NULL)
    fclose (fp);
}
```

EMSページが，いっぱいになったら次のEMSページをマップする

1行ずつ(fgets)により，テキスト内容を読み込む．fgetsからエラーが返ってきたら読み込み終了とする

EMSのページ番号(_page)と，ページ・フレーム内へのポインタ(wrk_ptr)を，テキスト各行のポインタ配列(_str_ptr)に代入する

1画面を表示する前に，エスケープ・シーケンスによって画面を消去する

1画面分の表示ループcur_lineを先頭として，1ページ分表示する．このため，cur_lineが20変化すると，20行スクロールしたのと同じになる

表示行のあるEMSページが，現在マップされていないEMSページであれば，表示行のあるページをマップする

入力キーによって動作を決める

ESCキーの場合は終わる

CRまたはスペース・キーであれば1行先へ進める

BSの場合は1行前に戻る

+が入力されたら，LINES_SKIP分だけ表示行を先へ進める(前方スクロール)

−が入力されたら，LINES_SKIPの分だけ，表示行を前へ戻す(後方スクロール)

第5章 2バイト・コードの処理は案外複雑

PC9801のIO.SYSの解析

MS-DOSのシステム・ファイルは二つあり，大半の仕事をするのがMSDOS.SYSです．機種ごとに異なる周辺装置の差を吸収するための基本デバイス・ドライバはIO.SYSにまとめられています．

斉藤 健司

ここでは，実際の**IO.SYS**をのぞいてみることにしましょう［PC9801用MS-DOS Ver.3.1(PS98-125.ほかにPS98-127/129/011がある)］.

いちばんてっとり早い方法は，**SYMDEB**などのデバッガを使って，インストール済みの**IO.SYS**を逆アセンブルすることでしょう．まず，**IO.SYS**がどこにロードされているのかを知らなければなりません．MS-DOSは，システムを下位アドレスに置き，アプリケーション・ソフトウェア用などユーザ領域のためにその後の上位アドレスを連続して有効に使えるようにしますので，**IO.SYS**も下位アドレスに置かれているはずです．

サーチ・コマンドを使い，CONやPRNの文字列を検索するといくつか見つけてくれます．このうちのどれかがデバイス・ドライバのデバイス・ヘッダ部分になっています．ディスプレイ・コマンドでメモリ・ダンプすると，前述した構造になっていますのでだいたい見当がつきます(図1)．

● デバイス・ヘッダ

デバイス・ヘッダを翻訳したのが**表1**です．このデバイス・ドライバは，標準でCON, AUX, PRN, CLOCK, ディスクの五つのデバイスをサポートしているのがわかります．

ストラテジおよびインタラプト(割り込み)ルーチンを逆アセンブルしたのが**リスト1**です．各デバイス・ヘッダのストラテジ・エントリ・ポイントは，すべてのデバイスで同じ値が記述されています．ですから，ここでサポートされるデバイスをシステムが利用する際には，どのデバイスでも同一のリクエスト・ヘッダ用セーブ・アドレスが使われます．

● コマンド・テーブル

インタラプト・ルーチンでは，各デバイス・ドライバの入口部分で，SIレジスタにコマンド・テーブルの先頭アドレスをセットして，共通ルーチンにジャンプしています．

インタラプト・ルーチン内部では，要求された(コマンド番号)×2＋SIによって，コマンド・テーブルから各コマンド処理ルーチンへのポインタを得て，ジャンプします．コマンド・テーブルをわかりやすく**表2**に示しましたので参照してください．

従って，コマンド・テーブルの内容はコマンド処理ルーチンへのポインタということになりますから，こ

〈図1〉[8] IO.SYS(V3.1)のデバイス・ヘッダ部分のダンプ・リスト

：リンク・ポインタ，
：デバイス名

```
-D27D0 28FF
0060:27D0  20 00 00 90 E6 27 60 00-13 80 FA 28 05 29 43 4F   ...f'`....z(.)CO
0060:27E0  4E 20 20 20 20 20 F8 27-60 00 00 80 FA 28 0B 29  N     x'`....z(.)
0060:27F0  41 55 58 20 20 20 20 20-0A 28 60 00 00 80 FA 28  AUX     .(`....z(
0060:2800  11 29 50 52 4E 20 20 20-20 20 1C 28 60 00 08 80  .)PRN     .(`....
0060:2810  FA 28 17 29 43 4C 4F 43-4B 20 20 20 FF FF FF FF  z(.)CLOCK   ....
0060:2820  00 28 FA 28 1D 29 02 90-5A 29 5A 29 5A 29 54 29  .(z(.)..Z)Z)Z)T)
0060:2830  6E 29 89 29 5A 29 A8 29-BC 29 BC 29 E9 29 5A 29  n).)Z)()<)<)i)Z)
0060:2840  5A 29 5A 29 5A 29 5A 29-5A 29 5A 29 5A 29 5A 29  Z)Z)Z)Z)Z)Z)Z)Z)
0060:2850  54 29 10 2A 2C 2A 5A 29-4D 2A 62 2A 62 2A 90 2A  T).*,*Z)M*b*b*.*
0060:2860  5A 29 5A 29 5A 29 5A 29-5A 29 5A 29 5A 29 5A 29  Z)Z)Z)Z)Z)Z)Z)Z)
0060:2870  5A 29 54 29 5A 29 50 29-5A 29 5A 29 DF 2A DF 2A  Z)T)Z)P)Z)Z)_*_*
0060:2880  0D 2B 5A 29 5A 29 5A 29-5A 29 5A 29 B7 2A 5A 29  .+Z)Z)Z)Z)Z)7*Z)
0060:2890  5A 29 5A 29 54 29 34 2B-50 29 5A 29 5A 29 41 2B  Z)Z)T)4+P)Z)Z)A+
0060:28A0  41 2B 5A 29 5A 29 5A 29-5A 29 5A 29 5A 29 5A 29  A+Z)Z)Z)Z)Z)Z)Z)
0060:28B0  5A 29 12 45 AB 45 14 46-54 29 1C 48 50 29 5A 29  Z).E+E.FT).HP)Z)
0060:28C0  5A 29 44 48 54 48 5A 29-5A 29 5A 29 E4 45 F0 45  Z)DHTHZ)Z)Z)dEpE
0060:28D0  01 46 5A 29 3A 03 8D 06-53 2D D5 40 E6 2B E2 40  .FZ):...S-U@f+b@
0060:28E0  97 2D 1B 41 AE 31 61 3E-C7 40 94 2B AA 31 4F 3E  .-.A.1a>G@.+*1O>
0060:28F0  03 41 95 2B 57 00 00 00-00 02 2E 89 1E D4 28 2E  .A.+W.........T(.
->CON
```

デバイス・ヘッダ

コマンド・テーブル

れをもとにポインタが指すルーチンを解析することによって，各コマンドがどのような働きをしているのかわかるというわけです．

なお，コマンド・テーブルの内容が，すべて目的処理ルーチンをさしているのではないことに注目してください．すなわち，システムの要求したコマンドが必ずしも目的のI/O処理に適合しないことがあるということです．例えば，入力デバイスに対して出力要求が発行されたり，サポートされていないコマンド要求が起こることもあるからなのです．

このような時，デバイス・ドライバはエラーと判断したり，エラーではないが何もせずに，エラーを含んだステータス情報をもって，システムに制御を返してやらなければなりません．これは，たんにコマンド・テーブルにエラー処理ルーチンへのポインタ，またはリターン処理のアドレスを記述することによって可能です．

出力処理ルーチンで行われていること

本稿はコンパチブル・マシンの製作が目的ではありませんので，システムからの要求による各デバイスの各処理ルーチンについて事細かく説明しても意味がありませんし，そのような誌面の余裕もありません．よ

〈表1〉IO.SYS(V3.1)のデバイス・ヘッダ

アドレス	リンク・ポインタ	タイプ	ストラテジ	インタラプト	デバイス名*
0060:27D4	0060:27E6	8013	28FA	2905	CON_____
0060:27E6	0060:27F8	8000	28FA	290B	AUX_____
0060:27F8	0060:280A	8000	28FA	2911	PRN_____
0060:280A	0060:281C	8008	28FA	2917	CLOCK___
0060:281C	FFFF FFFF (−1)	2800	28FA	291D	(n)**

＊　デバイス名の後の“_”は，スペースを表す．
＊＊ nはブロック・タイプ・デバイスの数

〈表2〉IO.SYS(V3.1)コマンド・テーブルの内容

コマンド	CON	AUX	PRN	CLOCK	(BLOCK)
0	295A	295A	295A	295A	4512
1	295A	295A	295A	295A	45AB
2	295A	295A	295A	295A	4614
3	295A	295A	295A	295A	2954*
4	295A	2A10	295A	2B34	481C
5	295A	2A2C	2950**	2950**	2950**
6	296E	295A	295A	295A	295A
7	2989	2A4D	295A	295A	295A
8	29A8	2A62	2ADF	2B41	4844
9	29BC	2A62	2ADF	2B41	4854
10	29BC	2A90	2B0D	295A	295A
11	29E9	295A	295A	295A	295A
12	295A	295A	295A	295A	295A
13	295A	295A	295A	295A	45E4
14	295A	295A	295A	295A	45F0
15	295A	295A	295A	295A	4601
16	295A	295A	2ADF	295A	295A

* done，busy をセット
** error，done をセット
処理ルーチンへのポインタ

〈リスト1〉[8] ストラテジ・ルーチンとインタラプト・ルーチンの逆アセンブル・リスト(IO.SYS V3.1)

```
-U28FA 296D                                  ストラテジ・エントリ
0060:28FA 2E891ED428      MOV     CS:[28D4],BX      リクエスト・ヘッダへのポインタ(BX)をセーブ
0060:28FF 2E8C06D628      MOV     CS:[28D6],ES
0060:2904 CB              RETF                      MS-DOSに制御を戻す
                                                    リクエスト・ヘッダへのポインタ(ES)をセーブ
                                             インタラプタ・エントリ
0060:2905 56              PUSH    SI
0060:2906 BE2828          MOV     SI,2828           CONエントリ
0060:2909 EB16            JMP     2921
0060:290B 56              PUSH    SI                AUXエントリ
0060:290C BE4A28          MOV     SI,284A
0060:290F EB10            JMP     2921
0060:2911 56              PUSH    SI                PRNエントリ
0060:2912 BE6C28          MOV     SI,286C
0060:2915 EB0A            JMP     2921
0060:2917 56              PUSH    SI                CLOCKエントリ
0060:2918 BE8E28          MOV     SI,288E
0060:291B EB04            JMP     2921
0060:291D 56              PUSH    SI                ブロック・デバイス・エントリ
0060:291E BEB228          MOV     SI,28B2
0060:2921 50              PUSH    AX
0060:2922 51              PUSH    CX
0060:2923 52              PUSH    DX
0060:2924 57              PUSH    DI                リクエスト・ヘッダへのポインタをDS:BXにロード
0060:2925 55              PUSH    BP
0060:2926 1E              PUSH    DS
0060:2927 06              PUSH    ES
0060:2928 53              PUSH    BX
0060:2929 2EC51ED428      LDS     BX,CS:[28D4]
0060:292E 8A4701          MOV     AL,[BX+01]
0060:2931 8A670D          MOV     AH,[BX+0D]        コマンド・コードをALに
0060:2934 8B4F12          MOV     CX,[BX+12]
0060:2937 8B5714          MOV     DX,[BX+14]
0060:293A 97              XCHG    AX,DI
0060:293B 8A4702          MOV     AL,[BX+02]
0060:293E 32E4            XOR     AH,AH
0060:2940 03F0            ADD     SI,AX             コマンド・テーブルのアドレス計算
0060:2942 03F0            ADD     SI,AX
0060:2944 3C10            CMP     AL,10             不正コマンドの確認
0060:2946 770C            JA      2954
0060:2948 97              XCHG    AX,DI             DS＝CSとする
0060:2949 C47F0E          LES     DI,[BX+0E]
0060:294C 0E              PUSH    CS
0060:294D 1F              POP     DS
0060:294E FF24            JMP     [SI]              目的I/O処理ルーチンへジャンプ
0060:2950 B403            MOV     AH,03
0060:2952 EB08            JMP     295C              DONE,BUSYをセット
0060:2954 B003            MOV     AL,03
0060:2956 B481            MOV     AH,81             デバイス・エラー(Unknown Command)セット
0060:2958 EB02            JMP     295C
0060:295A B401            MOV     AH,01
0060:295C 2EC51ED428      LDS     BX,CS:[28D4]
0060:2961 894703          MOV     [BX+03],AX        ERROR,DONEをセット
0060:2964 5B              POP     BX
0060:2965 07              POP     ES
0060:2966 1F              POP     DS
0060:2967 5D              POP     BP
0060:2968 5F              POP     DI
0060:2969 5A              POP     DX                DONEセット(正常終了)
0060:296A 59              POP     CX
0060:296B 58              POP     AX                リクエスト・ヘッダにステータス・コードを書き込む
0060:296C 5E              POP     SI
0060:296D CB              RETF                      MS-DOSへ制御を戻す
-> CON
```

って，ここでは一例としてPRNの出力処理に的をしぼって検討してみましょう．

● 2バイト・コードの処理はめんどう

1バイト処理系(欧米のシステム)では，**リスト2**のように，1文字入出力の際，気兼ねなくBIOSをコールし，即リターンします．しかし，2バイト圏(漢字コード)である日本では，そうはいきません．プリンタ出力要求が発生したとき，システムが送ってくるコードは，ANKとシフトJISコードが混在しており，これを単純にプリンタに送り出しても，目的の印字は行われません．これは，プリンタ側がシフトJISを認識しないためです．

通常，漢字印字が必要な場合には，まずエスケープ・シーケンスをプリンタに送り漢字モードにし，漢字コードが認識できるようにしてからでないとうまくいきません．しかも，漢字コードはJISコードでなければなりません．もちろん，プリンタが漢字モード中にANK印字要求が発生したときは，漢字モードを解除してからANK文字を送り出します．

● 漢字の処理には手間がかかる

このデバイス・ドライバの漢字処理部分の逆アセンブル・リストは長いので，解析したフローチャートを**図2**に示します．漢字処理つきの1文字出力を実行するために，1バイト圏のマシンに比較して，手間のかかることを行っているのが，おわかりいただけると思います．

シフトJIS対応のプリンタは見たことがありませんが，もし存在するならデバイス・ドライバにこれだけめんどうな漢字処理をもつ必要はありませんし，現在のところプリンタ・コード体系が統一されていませんので，MS-DOS専用プリンタとしての存在意義はあります．というのも，システムが漢字処理分だけ軽くなりますので，I/O処理から早く抜け出て，結果的に本来の処理にCPUタイムを振り向けることができるからです．

システムが内部処理型式そのままの状態で，I/Oにデータを渡してやることによって，スループットが向上します．プリンタが，いかに大きなプリント・バッファをもっていても，前述した処理は通過しなければなりませんので，CPUはI/Oでもたつくことになります．

〈リスト2〉[9] PC-DOS 3.3 PRN-OUT部分の逆アセンブル・リスト(IBMBIO.COM V 3.3)

```
 -u07e6 07ff
0070:07E6 E310          JCXZ    07F8        ← 出力キャラクタをALに
0070:07E8 BB0200        MOV     BX,0002
0070:07EB 268A05        MOV     AL,ES:[DI]
0070:07EE 47            INC     DI
0070:07EF 32E4          XOR     AH,AH       ← AH=0→Print Character
0070:07F1 E82400        CALL    0818
0070:07F4 7505          JNZ     07FB
0070:07F6 E2F0          LOOP    07E8
0070:07F8 E98DFE        JMP     0688
0070:07FB 4F            DEC     DI
0070:07FC 4B            DEC     BX
0070:07FD 75EC          JNZ     07EB
0070:07FF E971FE        JMP     0673        ← IBMのプリンタBIOSコール
-u0818 0831
0070:0818 E891FE        CALL    06AC
0070:081B CD17          INT     17
0070:081D F6C408        TEST    AH,08       ← I/Oエラー？
0070:0820 740A          JZ      082C
0070:0822 B009          MOV     AL,09
0070:0824 F6C420        TEST    AH,20       ← 紙なし検出？
0070:0827 7502          JNZ     082B
0070:0829 FEC0          INC     AL
0070:082B C3            RET
0070:082C B002          MOV     AL,02
0070:082E F6C401        TEST    AH,01       ← タイム・アウト?
0070:0831 C3            RET
->con
 -u6ac
0070:06AC 2E8B166C01    MOV     DX,CS:[016C]
0070:06B1 C3            RET
0070:06B2 E306          JCXZ    06BA
0070:06B4 E80600        CALL    06BD
0070:06B7 AA            STOSB
0070:06B8 E2FA          LOOP    06B4
0070:06BA E9CBFF        JMP     0688
0070:06BD 33C0          XOR     AX,AX
->con
```

デバイス・ドライバの常駐を知るツール

本文では，デバイス・ドライバの位置を知るためにデバッガを使っていますが，簡単にそれらの情報を表示してくれるのが，**dev.com**です．実行結果を**リストA**に示します．このツールはアスキー社の「MS-DOS SOFTWARE TOOLS-3」に入っています．

またメガソフト社のMEMORY-PRO386の中の**LU**コマンドでも知ることができます．

〈リストA〉デバイス・ドライバの常駐状況を表示

```
0906:0048 12EC 12F2 8004 NUL      (nul)
19B2:0000 0066 0071 8013 CON      (special) (stdout) (stdin)
1198:0000 0045 0050 A040 PRN      (output-until-busy) (generic ioctl)
1190:0000 0049 004E 0000 I:
1186:0000 005B 0066 C000 VEMMXXX0 (ioctl)
1044:0000 0045 0050 A040 PRN      (output-until-busy) (generic ioctl)
0060:3400 354E 3597 8013 CON      (special) (stdout) (stdin)
0060:3412 354E 359D 8000 PRN
0060:3424 354E 35A3 8000 AUX
0060:3436 354E 35A9 8008 CLOCK    (clock)
0060:3448 354E 35AF 4840 A: - H:  (ioctl) (open/close/rm)
```

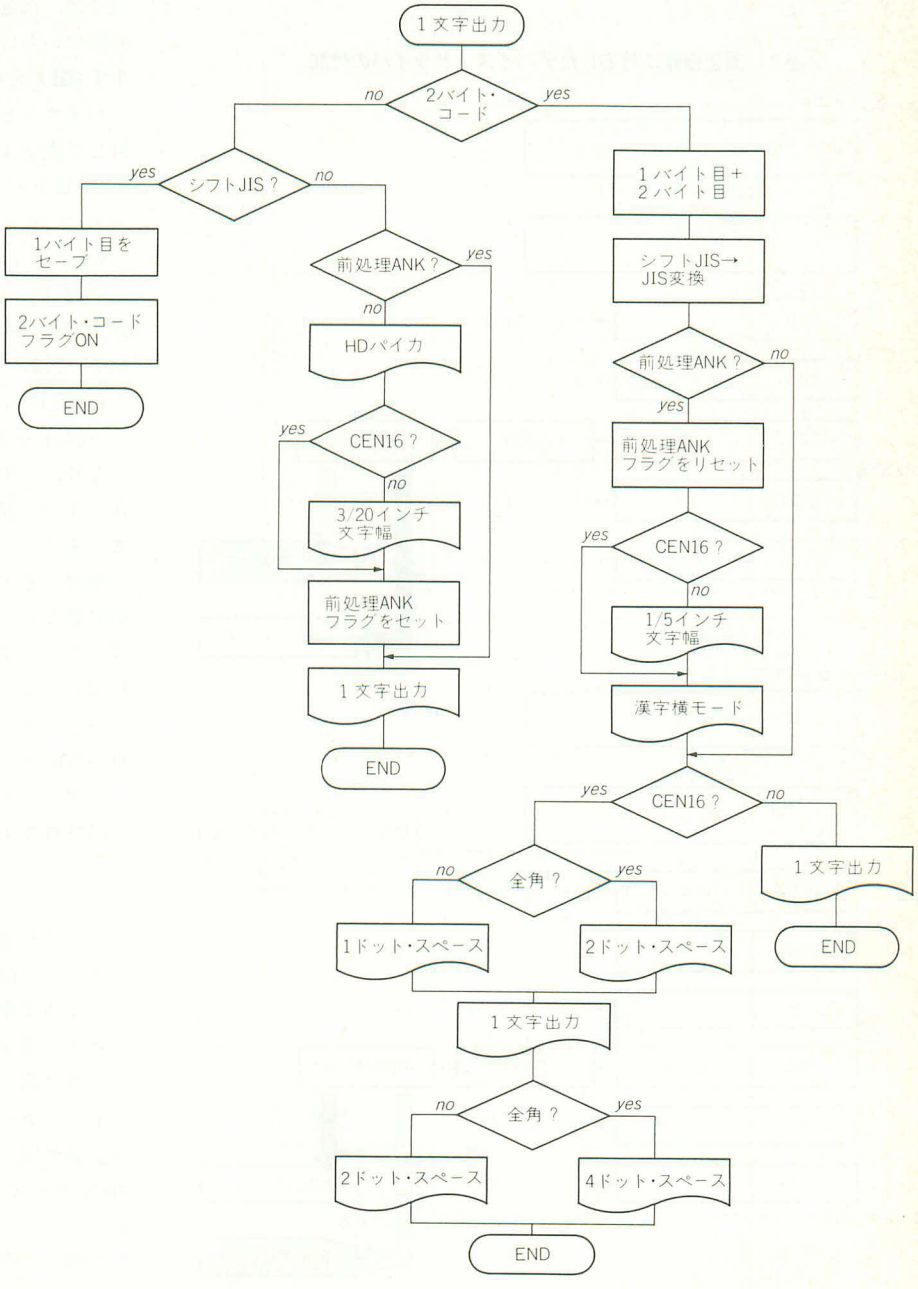
互換性のためのデバイス・ドライバ

しかし，シフトJIS→JIS変換処理を見ると，逆の視点からデバイス・ドライバの利点を再認識することができます．すなわち，目的ハードウェアがどのようなものであっても，インターフェースを取ることが可能であるという点です(コード体系やハードウェアの違いを吸収できる)．

一般的にパソコンのシステム・メーカは，自社のI/Oを標準と考えます．ところが，ほかのメーカの周辺機器の接続が必要になった場合，周辺機器に対応したデバイス・ドライバを付加するか，システム供給側とコンパチブルになるように周辺機器の変更が必要になります．新規に周辺機器を作成する場合は後者でもいいのですが，既存の周辺機器に対応したい場合には，デバイス・ドライバによるI/Oサポートのほうが柔軟に対応できます．もともとデバイス・ドライバの目的はここにあるわけです．

たとえば，**図3**のようにコード体系が異なっているため他社製のプリンタが使えず，お倉のゴミと化していることがよくあります．しかし，これに対応するデバイス・ドライバを作成することによって，このプリンタは見事に社会復帰することが可能です．

〈図2〉
IO.SYS(V3.1) PRN
出力漢字処理部



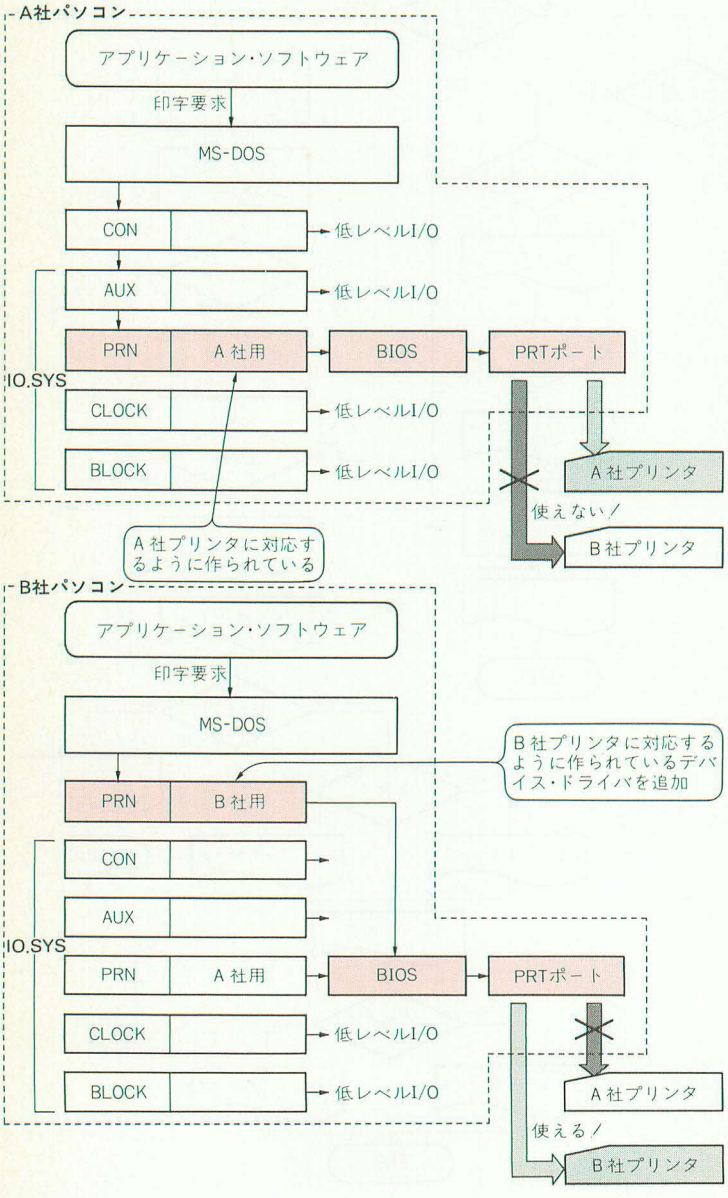
筆者は，専用のプリンタ・デバイス・ドライバを作成し，PC9801に富士通製の古いプリンタ(MB27410)を接続して，ばら打ち帳票印刷業務に使っていました．

● すべてのプログラムがデバイス・ドライバを経由するとはかぎらない

ただし，これはあくまでアプリケーション・ソフトウェアやユーティリティ・ソフトウェアが，階層構造に従って，OSに対して目的ハードウェアのI/O処理を要求したときの利点です．表皮に存在するアプリケーション・ソフトウェアが，I/O順位を無視してより下位にあるBIOSコールを行ったり，独自のI/Oハンドラ・プログラムをもつ場合があります．

こうなると，せっかく作成したデバイス・ドライバもまったく役に立ちません．

〈図3〉周辺機器に対応したデバイス・ドライバの付加



もうひとつ考えなければならないこと

日本語の文章は各行の左右端を合わせるのが普通で，全角，半角という文字サイズの問題があります．

● 漢字とANKのピッチは1：2の比率

ほとんどのプリンタは，ANKの場合，デフォルトでパイカ・ピッチが選択され，これが最もフレンドリ(プログラム・リストなどでおなじみ)なわけですが，パイカ・ピッチでANK 2文字の幅に漢字1文字(全角)が割り当てられるようにしないと，ANKおよび漢字混在の印字結果は右端がばらばらになり，読みにくくなってしまいます．したがって，このデバイス・ドライバでは，前述した漢字コードの出力の際にいちいちこの調整を行っています(図2を参照)．

パイカ・ピッチのANK 2文字に対して漢字1文字の幅は，

▶24ピン・プリンタ(160ドット/インチ)では，

(漢字ドット数)24＋(文字間隔)8＝32ドット

▶16ピン・プリンタ(120ドット/インチ)では，

(漢字ドット数)18＋(文字間隔)6＝24ドット

となり，いずれもANK文字間隔は0ですが，漢字文字間隔は開いてしまいます．

単純に考えると，漢字の字間ゼロが必要なら，エスケープ・シーケンスによって文字幅を強制的に設定すればいいような錯覚をおぼえます．デバイス・ドライバは，制御コードをANKとして扱いますのでトランスペアレント(透過)にプリンタへ送り出されますが，次に文字間ゼロのつもりで送った漢字コードはプリンタへ送り出されるまえにいったんトラップされます．

ここで，問題のデバイス・ドライバによる文字間設定制御コードが割り込み，漢字コードより先にプリンタへ送り出されるというわけです．

次に，アプリケーション・ソフトウェアでは，これまで説明してきたPRNデバイス・ドライバを経由してプリント・アウトしているかどうかを調べてみます．

アプリケーション・ソフトウェアのI/Oアクセス

ここでは，実際のアプリケーション・ソフトウェアやユーティリティ・ソフトウェアが，どのようにI/Oをアクセスしているのか調べてみます．デバイス・ドライバでも具体例としてPRT処理を検証したので，アプリケーション・ソフトウェアの印字処理について，いろいろと話題が多く有名な一太郎(Ver 3)君をレントゲン台にのせて見ていきましょう．

OS-9のデバイス・ドライバ

MS-DOSが手本としたOSが，UNIXであるということは広く知られています．パソコンにその環境をもってくる際に，MS-DOSは，UNIXのもつ多くの特徴をそぎ落としました．その結果，シングル・タスク/シングル・ユーザのOSとなり，ファイル管理の方法とI/Oの階層構造をデフォルメした型式を踏襲しています．

ところが，それ以前にUNIXの環境を8ビット・パソコンに移植しようとした，別のOSがありました．いわずと知れたOS-9ですが，こちらは，MS-DOSに比べると，UNIXから多くを引き継いでおり，マルチ・タスク/マルチ・ユーザをサポートしています．

マルチ・タスク/マルチ・ユーザをサポートするためには，OSの管理が厳しくなります．また，雑多なプログラムによって次々に発生するI/O処理に対応しなければなりませんので，デバイス・ドライバの書式が，MS-DOSのそれ以上に厳格に定義されています．また，図AのようにI/O処理部門が，MS-DOSデバイス・ドライバのように一元化されているのではなく，さらに階層化され目的別のブロックによって構成されています．

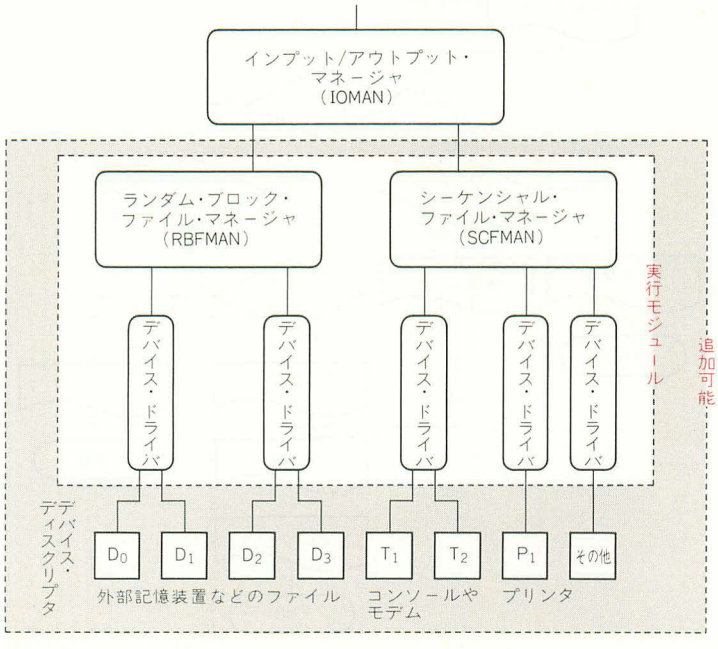
このうち，IOMANはコアに密着した，いわばシステムの一部と解釈できるものです．次のファイル・マネージャは，OS-9の互換性保持に重要な役割をもつ実行モジュール(リエントラント)です．すなわち，個々のデバイスの違いを吸収して，上位のシステムとの同一型式のデータ・フローを形成します．ですから，ファイル・マネージャの下位には，いろいろなデバイスがぶら下がるというわけで，その下がデバイス・ドライバとなります．

しかし，MS-DOSのデバイス・ドライバと違って，この下にBIOSがあったりしません．ここが実行モジュールとして最下位レベルで，実際にI/Oデバイスの制御，入出力を行うI/Oハンドラ(リエントラント)なのです．ところが，デバイス名やI/Oデバイスの物理アドレスなどの定数は，その下にあるデバイス・ディスクリプタと呼ばれるテーブル・モジュールに記述します．

これによって，同一種類のI/Oデバイスがいくつあっても，これを制御するデバイス・ドライバが1本ですむことになります．したがって，たとえば，現行のものと同じ形のI/Oデバイスを増設しても，デバイス・ドライバを書き足すのではなく，新規分のデバイス・ディスクリプタを付加するだけとなります．これはマルチ・ユーザをサポートするために生まれた必然性なのかもしません．

MS-DOSに比べて自由な記述は許されませんが，これによって，外部記憶装置のファイル，その他のI/O，ROMデータに至るまで，機種に依存しない互換性が保証されることになります．

〈図A〉OS-9のI/O階層構造



JXWの印刷メニュから印刷を始めると，要求した書式にしたがって目的の文書がプリント・アウトできます．この時，MS-DOSにファンクション・コール(**INT 21H**)を使ってアクセスしているのならば，絶対にPRTのデバイス・ドライバを通過するはずです．

検証の最も簡単な方法はMS-DOS Ver.3.3を使って，**DEVICE＝PRINT．SYS**を**CONFIG．SYS**から外して立ち上げます．また，**IO．SYS**にPRTドライバが標準で乗っかっているバージョンでは，SYMDEBを使って，PRTへのリンク・ポインタをはずしてやるのです．**リスト3**のようにすると，PRTがシステムに存在しなくなります．ここから，JXWを起動してサンプル印字を試みてみます．

● ワープロではBIOSを直接呼ぶ

いかがでしょうか？PRTデバイス・ドライバにはまったく無関係に目的の印刷が可能です．結果的には，

MS-DOS 3.3のPRINT.SYS

MS-DOS 3.3では**PRINT．SYS**が，**IO．SYS**から分離され，内容も多少変わっています．

まず，**SWITCH．COM**により，1/5インチ文字幅設定(ANK：漢字＝1：2)と文字間隔ゼロ(2：3)のふたとおりのモードがサポートされています．これは，絶対アドレス**0060:1E57**(9801用の**IO．SYS**は通常セグメント**0060**にロードされる)のビット0が，**"0"**のとき前者の設定，**"1"**のとき後者の設定となります．

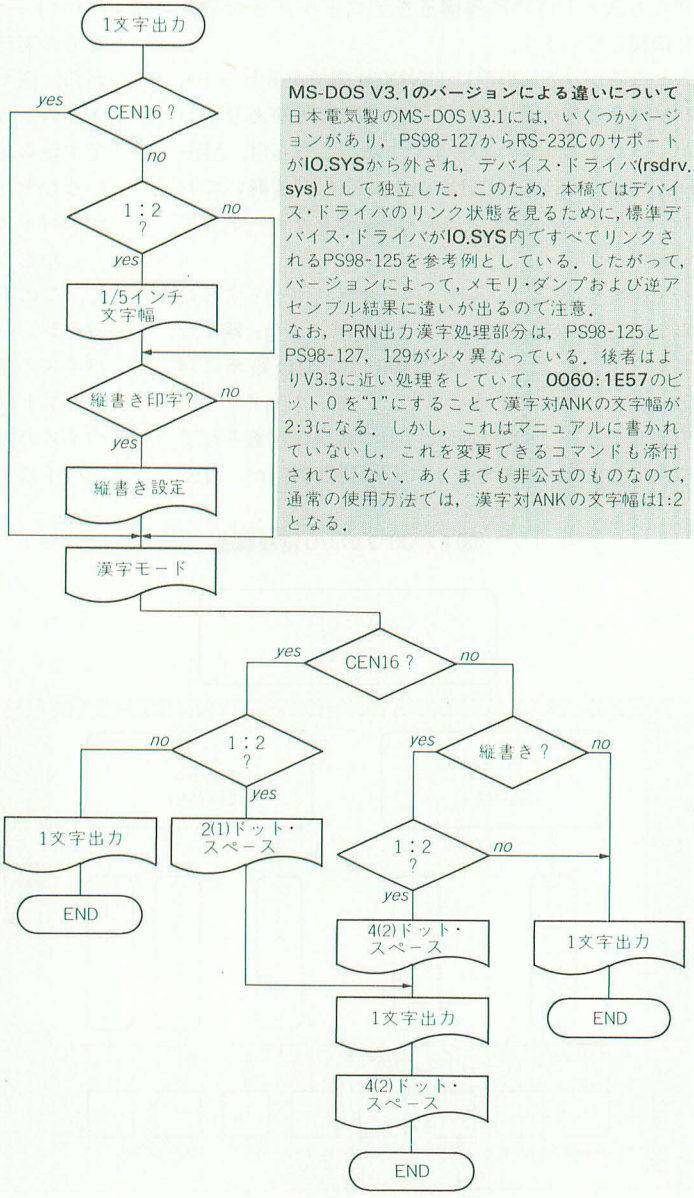
ですから，後者の文字間ゼロの設定は，デバイス・ドライバが文字幅を規定せず，シフトJIS→JISの変換以外は，トランスペアレントにプリンタに送出しますので，ユーザ・プログラムから文字間隔のコントロールが自由に可能になります．

それから，**0060:1E57**のビット1の値を**"1"**にすることで，たて書き印字になります．ただし，1/5インチ文字幅設定時には，これに加えて8ドット・スペーシングを行いますので，さらに文字間が開いてしまいます．

なお，**0060:008D**の内容は，Ver 3.1と同様にイニシャライズの際に，メモリ・スイッチ5の内容をコピーしたもので，ビット0が，CEN 16, 24を表しています．

PRINT．SYS(Ver 3.3)の漢字処理部のフローチャートを図Bに示します．

〈図B〉PRINT．SYS(V 3.3)漢字処理部



MS-DOS V3.1のバージョンによる違いについて

日本電気製のMS-DOS V3.1には，いくつかバージョンがあり，PS98-127からRS-232Cのサポートが**IO.SYS**から外され，デバイス・ドライバ(**rsdrv.sys**)として独立した．このため，本稿ではデバイス・ドライバのリンク状態を見るために，標準デバイス・ドライバが**IO.SYS**内ですべてリンクされるPS98-125を参考例としている．したがって，バージョンによって，メモリ・ダンプおよび逆アセンブル結果に違いが出るので注意．

なお，PRN出力漢字処理部分は，PS98-125とPS98-127，129が少々異なっている．後者はよりV3.3に近い処理をしていて，**0060:1E57**のビット0を"1"にすることで漢字対ANKの文字幅が2:3になる．しかし，これはマニュアルに書かれていないし，これを変更できるコマンドも添付されていない．あくまでも非公式のものなので，通常の使用方法では，漢字対ANKの文字幅は1:2となる．

プリンタBIOS(**INT 1AH**)を無効にすると印刷ができなくなり，何もせずに元の画面に戻ってしまいます．

ですから，一太郎というアプリケーション・ソフトウェアは，プリント・アウトについていうと**MS-DOS**ファンクション・コールを利用せず，最下位レベルの**BIOS**によってI/Oアクセスが行われていることになります．

● MS-DOSのプリンタ用ドライバでは漢字の字間があく

デバイス・ドライバの解析のところで説明したように，MS-DOSがパイカ・ピッチを優先的に扱っており，漢字文字幅がANK文字幅によって影響を受けてしまっています．このため，(ここは，推測になってしまいますが)本アプリケーション・ソフトウェアの作成者は，標準添付のデバイス・ドライバによる文字幅自動設定に不満があったと思われます(BIOSコールを選択したのが，すべて文字幅設定が原因だと述べているのではありません)．

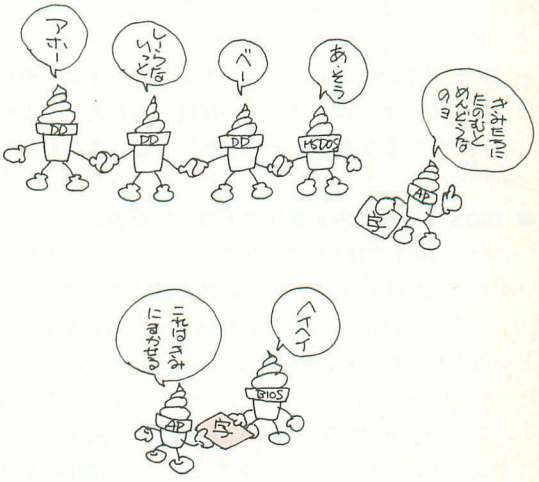
デバイス・ドライバ経由のプリンタ制御では，文字幅設定用の制御コードはプリンタへ確実に到達はするものの，漢字コード送出がトリガとなってMS-DOS方式の印字書式に切り替わり，ユーザ設定は徒労に終わることになります．これでは，プリンタを最大限に活用することは不可能ですし，いろいろな書式に対応することはできません．プログラム・リストなどANK文字が重要な意味をもつもののプリント・アウトでは，きれいな印字が可能ですが，漢字中心の日本語ワープロの印刷ではマイナス面が多くなります．まさに，小さな親切大きな迷惑，ですネ．

実際この要領で印字した漢字混じりの文章は漢字の字間が空いてしまいます．参考までにプリンタ送出データのHEXダンプ・リスト(**リスト4**)を示します．

〈リスト3〉PRNへのリンク・ポインタをはずす

```
-D27D0
0060:27D0  20 00 00 90 E6 27 60 00-13 80 FA 28 05 29 43 4F   ...f'`...z(.)CO
0060:27E0  4E 20 20 20 20 20 F8 27-60 00 00 80 FA 28 0B 29  N     x'`...z(.)
0060:27F0  41 55 58 20 20 20 20 20-0A 28 60 00 00 80 FA 28  AUX     .(`...z(
0060:2800  11 29 50 52 4E 20 20 20-20 20 1C 28 60 00 08 80  .)PRN     .(`...
0060:2810  FA 28 17 29 43 4C 4F 43-4B 20 20 20 FF FF FF FF  z(.)CLOCK    ....
0060:2820  00 28 FA 28 1D 29 02 90-5A 29 5A 29 5A 29 54 29  .(z(.)..Z)Z)Z)T)
0060:2830  6E 29 89 29 5A 29 A8 29-BC 29 BC 29 E9 29 5A 29  n).)Z)()<)<)i)Z)
0060:2840  5A 29 5A 29 5A 29 5A 29-5A 29 5A 29 5A 29 5A 29  Z)Z)Z)Z)Z)Z)Z)Z)
-E27E6
0060:27E6  F8.0A    27.28 ;Eコマンドで AUXのリンク・ポインタをCLOCKに変更
-D27D0
0060:27D0  20 00 00 90 E6 27 60 00-13 80 FA 28 05 29 43 4F   ...f'`...z(.)CO
0060:27E0  4E 20 20 20 20 20 0A 28-60 00 00 80 FA 28 0B 29  N     .(`...z(.)
0060:27F0  41 55 58 20 20 20 20 20-0A 28 60 00 00 80 FA 28  AUX     .(`...z(
0060:2800  11 29 50 52 4E 20 20 20-20 20 1C 28 60 00 08 80  .)PRN     .(`...
0060:2810  FA 28 17 29 43 4C 4F 43-4B 20 20 20 FF FF FF FF  z(.)CLOCK    ....
0060:2820  00 28 FA 28 1D 29 02 90-5A 29 5A 29 5A 29 54 29  .(z(.)..Z)Z)Z)T)
0060:2830  6E 29 89 29 5A 29 A8 29-BC 29 BC 29 E9 29 5A 29  n).)Z)()<)<)i)Z)
0060:2840  5A 29 5A 29 5A 29 5A 29-5A 29 5A 29 5A 29 5A 29  Z)Z)Z)Z)Z)Z)Z)Z)
->CON
```

JXWのプリント出力



I/Oアクセス・レベルと互換性

MS-DOS上のプログラムは，すべてCPUの制御権

〈リスト4〉
実際の印字とそのプリントHEXダンプ

```
出力文字
Device Driver は，有用である．

建設的なI/O Access の閃き．

16進ダンプ・モード
  44 65 76 69 63 65 20 44 72 69 76 65 72 20 1C 42   Device Driver  B
  1B 4B 24 4F 21 24 4D 2D 4D 51 24 47 24 22 24 6B   .K$O!$M-MQ$G$"$k
  21 25 1B 48 1C 41 0D 0A 0D 0A 1C 42 1B 4B 37 7A   !%.H.A.....B K7z
  40 5F 45 2A 24 4A 1B 48 1C 41 49 2F 4F 20 41 63   @_E*$J.H.AI/O Ac
  63 65 73 73 20 1C 42 1B 4B 24 4E 41 2E 24 2D 21   cess .B.K$NA.$-!
  25 1B 48 1C 41 0D 0A                              %.H.A..
▶上段が，実際の印字見本．
▶HEXダンプを見ると，漢字モード(ESC "K")にする前に，FS "B"を送っているのがわかる．
▶ESC "H"でANKが，HDパイカ・モードになる．
```

を得るとどのレベルのI/O処理も自由に活用することができます(図4)．これは，情報工学的OSの定義からはあまり薦められることではなく，ハードウェア，ソフトウェア資源の有効活用，という大命題がもろくも崩壊してしまうのです．

システムがある時間単位でしかも優先順位付きでI/Oアクセスを管理しているマルチタスク・システムでは絶対に認められない，下剋上的，反動分子的，ガキ大将的カオスが，MS-DOSシステムで横行しているというわけです．

したがって，MS-DOSはOSの概念からはOSもどきと呼んだほうが適切かもしれません．現にディスク・オペレーティング・システムという控え目な名称からも，ディスク・ファイルを中心に管理するシステム，ファイル互換を最も大切なポリシに掲げている基本システム，としてとらえた方がベターです．

● BIOSを直接利用するソフトウェアが多い

しかし，現在ではOS供給側である，当のマイクロソフト社も，IBM系プログラミング言語ソフトウェアをはじめとして，BIOSなしでは快適な操作性，機能の向上が得られないと考えたのか，Quickシリーズなど，軒並BIOSコールを正々堂々と行っています．ですから，あるハードウェアが一般に普及して，図4のように市民権を得る(シェアを確保する)と，互換性の条件としてBIOSもOSと共に，アプリケーション・ソフトウェア作成時の検討項目の中に入ってきます．

国内ではPC9801，欧米諸国ではIBM PC(互換機を含む)用ソフトウェアは，一部ファイル操作を除いてほとんどのアプリケーション・ソフトウェアが，BIOSルーチンを借用しています．IBMの場合はオープン・アーキテクチャですから別にして，NECはPC9801のBIOSルーチンを一般に公表していませんが(ソフト関係の雑誌にはかなり詳しく掲載されている)，たいていのアプリケーション・ソフトウェアはBIOSルーチンを利用しています．

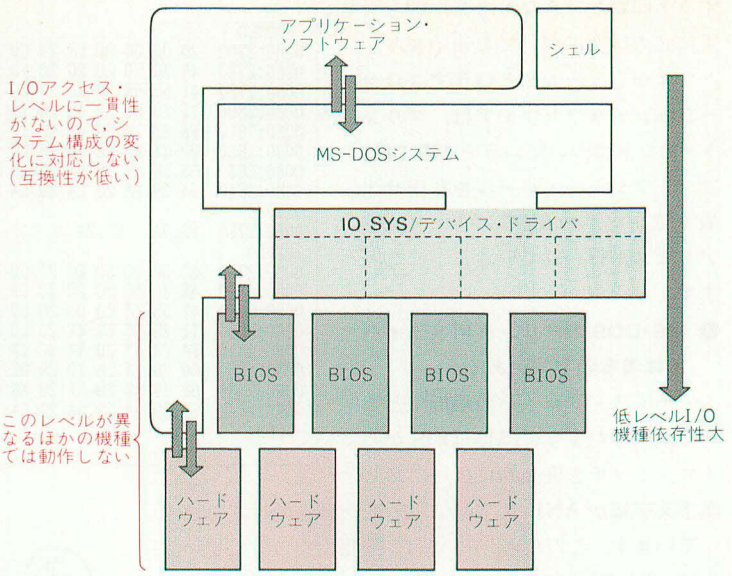
● BIOSはデバイス・ドライバのように簡単にすげかえられない

BIOSはROMに置かれるのが普通で，その機種に備わっているI/Oハードウェアのハンドラで，機種のモデル・チェンジがあってBIOSそのもののプログラムが変更されても，入出力パラメータはあまり変化しません．

しかし，パソコン・メーカもハードウェアや基本ソフトウェア(OS)など，周辺の環境の進歩に合わせていかなければなりませんので，旧態依然としたBIOSを永年に渡ってサポートする保証は，どこにもありません．

ですから，BIOSコールによるアプリケーション・ソフトウェアやユーティリティ・ソフトウェアは，ハードウェア・メーカの機種に依存したソフトウェアであることには変わりがないということです．もっとも，そのころには廃品種になっているのでしょうが．

〈図4〉互換性の低いI/Oアクセス



◆ 参考・引用*文献 ◆

(1) IBM Corporation, Disk Operating System Version 3.30 Reference, First Edition 80X0667, 1987 April.
(2) IBM Corporation, Disk Operating System Version 3.30 Technical Reference, First Edition 80X0945, 1987 April.
(3) Microsoft Corporation, Macro Assembler 5.0 Programmer's Guide, 1987.
(4) NEC，日本語MS-DOS 3.1 ユーザーズリファレンスマニュアル．
(5) NEC，日本語シリアルプリンタPC-PR201Gユーザーズマニュアル，1988.
(6) NEC，PRINT.SYS日本語MS-DOS 3.3.
(7) Seagatex, ST01 ROM-BIOS V 1.7, 1987.
(8)* NEC，IO.SYS日本語MS-DOS 3.3.
(9)* IBM，IBMBIO.COM PC-DOS 3.3.

第6章 漢字とANKの比率を1：2にしたままでパイカ文字以外でも使える プリンタ・ピッチを変えるユーティリティの作り方

MS-DOS上のプリンタ出力では文字がばらばらに出てきます．ANK文字種はパイカしか出せません．エリートなどが使えれば，1行にたくさんの文字を打つことができます．ここでは，IO.SYSの中のプリンタ・ドライバ部を直接変更して実現します．

中野 正次

PC9801のMS-DOS 3.1の最新バージョン以前のものでは，プリンタ・ドライバが**IO.SYS**に組み込まれています．このドライバは，MS-DOSの漢字コード(＝shift jis)をPC内部のコード(基本的にはjis)に変換し，8ビット文字と16ビット文字の切り替え信号をプリンタに送り出します．

このとき使われている制御コードがPC-PR×××用のコードなので，MS-DOSから直接(TYPEやDIRなどで)プリンタに打ち出せるのは，PC対応プリンタ(＝PC-PR×××用のコードを解釈できる)に限られるわけです．

もっとも，このことについては，純正以外のプリンタもPC対応になっているものが多いので大きな問題はありません．またワープロ・ソフトなどは，MS-DOSを経由しないで直接プリンタへ出力するようになっているので，やはりプリンタについての問題は少なくなっています．

● 文字ピッチはパイカが基本

プリンタ・ドライバについて，最も困るのは，汎用コンパイラを使って作られたアプリケーション・プログラムでプリンタを使いたい場合です．前述のMS-DOS内蔵のドライバでは，半角罫線や16ビットANK文字が使えず(8ビットANKに変換されてしまう)，また文字ピッチもパイカ(＝10 CPI)に限られます．

たとえば，プリンタ側の設定でエリート(12 CPI)にしておいても，16ビットの文字が入ると，その後のピッチはパイカにもどってしまいます．アプリケーションからESCシーケンスを使ってピッチ変更を行っても，同様の結果になります．

筆者の知るところでは，MS-DOSの標準テキスト・ファイルでも，プリンタに打ち出すためにワープロ・ソフトを使っている方が多いようです．また，簡単なアプリケーションをPascalを使って作成するように薦める場合にも，書式を整えたい場合は，そのソフトの中でプリンタのコントロールまでやらなければならず，本来のアプリケーションより手間がかかってしまうことになります．

● COPYキーでは罫線は印字されない

もっと困るのはCOPYキーによる画面のコピーです．CRT画面上には半角罫線も問題なく表示されるのに，コピーされたものには罫線は印字されません．画面コピーの機能はMS-DOSのドライバ中に含まれているので，アプリケーション側では解決できません．

いずれにしても，画面上のすべての文字がプリンタで印字でき，文字ピッチを変更してもその効果が持続し，さらに全角と半角が1：2のピッチになるようにできれば，印字を必要とするアプリケーションを組む場合やMS-DOSから直接印字する場合に極めて有効です．

プリンタ・ドライバの仕様

ところで，NECのMS-DOSのプリンタ・ドライバが何故パイカ・ピッチしかサポートしていないのでしょうか？ たんなる手抜きとも考えてみましたが，実際のところそれほど単純ではないようです．その理由は，ANKのピッチと漢字のピッチが1：2になるのはパイカの10 CPIのときだけで，ANKのエリート・ピッチは12 CPIなのに対して漢字は20/3(＝6.66)CPIとなっていて，このままでは1：2になりません．1：2になっていなくても使える用途はありますが，罫線のことも考えると，やはり一般的には必要度が高いといえます．

PC-PR×××のコントロール・コードには，1/160インチ単位のスペース送りが含まれていますが，このスペースを漢字1個あたり何個か入れる方法ではどうしてもうまくいきません．このことは，図1に示すとおりです．

このピッチを合わせることもかなりめんどうなのですが，問題はこれだけではありません．PC-PR×××系では，ANKと漢字の切り替えコードがピッチの指定も含んでいるのです．一方，MS-DOSの漢字コードはshift jisですから，プリンタ・ドライバのほうでANKと漢字の変化を検出して，その都度切り替えコードをプリンタへ送らなければなりません．このとき，

〈図1〉エリート・ピッチに漢字を合わせる計算

▶パイカの文字送り＝0.1インチ …… Ⓐ
▶漢字の5 CPI＝0.2インチ …… Ⓑ
Ⓐ×2＝Ⓑ
▶エリートの文字送り＝$\frac{1}{12}$インチ＝$\frac{10}{120}$インチ …… Ⓒ
▶漢字の20/3 CPI＝$\frac{3}{20}$インチ＝$\frac{18}{120}$インチ …… Ⓓ
Ⓒ×2−Ⓑ＝$\frac{2}{120}$＝$\frac{1}{60}$

漢字1文字当たり1/60インチを余分に送ればよい．しかし，スペース送りは1/160インチ単位なので，

$\frac{1}{60}$＝$\frac{8}{480}$＝$\frac{8}{160\times 3}$

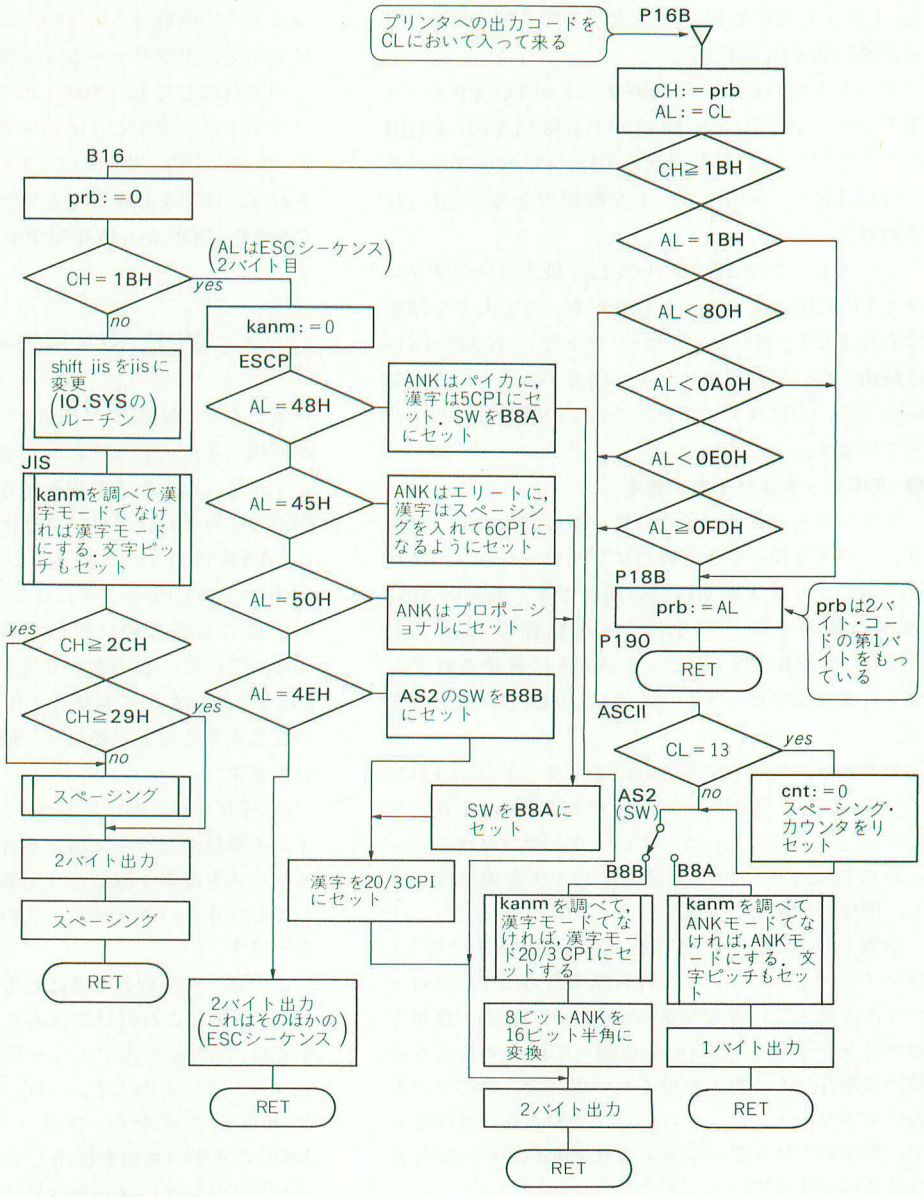
となって，漢字3文字当たり8ドットのスペースを送らなければならない計算になる．さらに半角16ビットの場合には，6文字当たり8ドットに切り替える必要がある．このためには，何らかのカウンタが必要になり，リターンのたびにリセットしなければならない．

ANKと漢字を切り替える専用のコードがあれば，ANK→漢字→ANKとなったときにもANKのピッチは元に戻りますが，ピッチの指定が含まれているので，ドライバが以前のANKのピッチを記憶していなければならないことになります．漢字のピッチについても同様です．

以上のように，パイカとエリートの両ピッチでANKと漢字の自動切り替えを行うだけでもかなりやっかいなことになります．これに，さらに欲ばって漢字の20/3 CPIに対応するANK 40/3 CPIも使えるようにすると，最大では10インチ幅の紙に114文字まで印字できるようになりますが，ドライバはもっと複雑になります．

しかし，どうせ作るならこれにプロポーショナル(可

〈図2〉
プリンタ・ドライバのフローチャート



変)ピッチも含めて4種類のピッチをESCシーケンスで指定できるドライバにしようということで，次のような仕様を作りました．

① ANKで10, 12, 13.3, プロポーショナルの4種類のピッチを1回のESCシーケンスで指定できる．
② プロポーショナル以外のピッチでは，漢字のピッチが自動的にANKの半分のCPIになる．
③ ANKと漢字が混在しても，一度指定したピッチは維持される．
④ 16ビット半角文字(罫線とANK)も印字でき，ピッチもずれない．
⑤ MS-DOSの**IO.SYS**を書き換えるプログラムとし，システム・エリアは1バイトも増えない．

このうち①～④までは特に困難はありませんが，これを元のドライバ(パイカ・ピッチのみ)と同じバイト数に収めるというのはかなり難しいと思われます．同じバイト数にしたい理由は，このドライバを使ったときに，以前から動いていたプログラムが走らなくなるような事態を避けるためです．同じバイト数になっていれば，メモリ量に関する問題は絶対に発生しませんから．

ドライバの設計

これまでの説明で，ドライバのもつべき機能は示されています．13.3 CPIの文字は，8ビットではなく16ビットの半角文字として印字しなければなりません．そのためには，8ビットのANKを16ビットに変換しなければなりませんが，スペース(**20H**)以外のコードは比較的簡単です．ただし，英数字とカナでは第1バイトが異なるので，この点だけは要注意です．このピッチのときはすべてが16ビット・コードなので，漢字とANKの切り替えは不要です．逆に切り替えコードを出さないようにしなければなりません．

● まずshift jisかESCコードかを判断

図2が設計されたドライバのフローチャートです．全体の流れとしては，MS-DOSへ渡される8ビットのコードがshift jisの1バイト目かまたはESCコード(=**1BH**)であれば次の2バイト目と合わせて処理し，それ以外は1バイトごとに処理するようになっています．このとき第1バイトを保持するのが**prb**です．

prbが**1BH**以上であるときに送られてきたものは，第2バイトと解釈されます．ESCシーケンスのうち，ドライバ内で解釈されるのは，第2バイトが**45H, 48H, 4EH, 50H**の4種類だけで，その他のものは手を加えずにプリンタへ送られます．したがって，ピッチ関係以外のすべてのESCシーケンスが使用可能です．

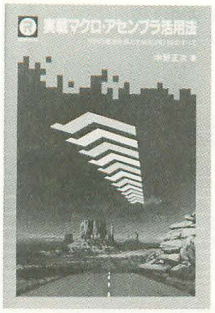
● ピッチの調整

12 CPIでの漢字スペーシングは，**cnt**をカウンタとしてモジュロ演算を行っています．10 CPIのときは，このルーチンは眠らせる必要がありますが，フローチャートには書いてありません．13.3 CPIのときは8ビットのコードを16ビットに変換するようにSWを切り替えています．これには，実際のプログラム中で分岐先を書き換える方式をとっています．このほかにも，プログラム中のコードを書き換える方法を使って，バイト数を節約しています．

実際のプログラムでは，**図2**のフローチャートのほかに，このコードを**IO.SYS**のしかるべき番地にロードする部分が必要になります．これを含めたリストが**リスト1**です(p.129)．ただし，このリストはNECのMS-DOS Ver 3.1(PS98-129)専用です．

ピッチ変更ユーティリティ

このドライバをユーザ・アプリケーションから使う



CQ RED BACKS

実戦マクロ・アセンブラ活用法

プロの要求を満たすMACRO 80のすべて

A 5判　288ページ　定価1,854円(税込)　中野正次 著

使い方によっては高級言語に劣らない能力を引き出すことも可能な，マクロ・アセンブラMACRO 80を例に，アセンブラのプログラム・テクニックを詳細に解説．

CQ出版社　〒170 東京都豊島区巣鴨1-14-2　☎03-947-6311　振替東京0-10665

には，ESC シーケンスでピッチ・セットを行えばよいのですが，できればコンソールからセットできたほうが便利です．ついでにドラフトと LQ(高速と高品位)の切り替えもできるようにしたのが**リスト 2** (Turbo Pascal)のプログラムです．

この中味は，押されたキーに対応したピッチと品位を ESC シーケンスとしてプリント・アウトするだけのもので，説明するほどではありません．このユーティリティを使用すると，MS-DOS で TYPE や DIR を実行して直接印字するときでも，7 種類のピッチ＊品位を利用可能になります．

図 3 がこのプリンタ・ドライバとユーティリティを使用して TYPE した印字サンプルです．プリンタは EPSON のものですが，PC 対応モードで使用しています．

同様のドライバを EPSON の ESC/P コード対応に作るとさらに 15 CPI の印字も可能になりますが，半角罫線が使えなくなるので一長一短です．半角罫線が使えなくなると画面のコピーも正しく行われないということになります．

〈リスト 2〉
文字種設定プログラム

V5.0ではこの行の(＊　＊)を消す

```
PROGRAM setfont ; (* IO.SYS のプリンタドライバにＦＯＮＴを指定する *)
(* New Version Including pica,eleet,propo,3/20 change control  *)
(* USES PRINTER ;   { This line must be active on T.P. V5.0 }  *)
TYPE w = STRING[2] ; w3 = STRING[3] ; stt = STRING[254] ;
CONST pk :w= #27#$48 ; el :w= #27#$45 ; pr :w= #27#$50 ; c3 :w= #27#$4E ;
 cr :w= #13#10 ; dr :w3 = #27#$64#$30 ; hh :w3 = #27#$64#$31 ;
VAR i,l,m : INTEGER ; chb : BYTE ;

BEGIN WRITE('プリンタドライバにＦＯＮＴを指定します。',cr,
'高速    パイカ    1                     高品位  パイカ      4',cr,
'高速    エリート  2                     高品位  エリート    5',cr,
'高速    半角      3                     高品位  半角        6',cr,
'    （中止        0)             高品位  プロポーショナル  7',cr) ;
REPEAT  WRITE(#13,'番号を指定してください。       ',#8,#8) ; READ(chb) ;
CASE chb OF
 1:WRITE(lst,pk,dr) ;
 2:WRITE(lst,el,dr) ;
 3:WRITE(lst,pk,dr,c3) ;
 4:WRITE(lst,pk,hh) ;
 5:WRITE(lst,el,hh) ;
 6:WRITE(lst,pk,hh,c3) ;
 7:WRITE(lst,pk,hh,pr) END  UNTIL chb<8 ;
IF chb=0 THEN WRITELN(cr,'変更しません。')

END .
```

〈図 3〉
7 種類の印字例

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

```
┌┐┬└┘┴││─
┌──┬──┐1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
│1234│漢字│11223344556677889900AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVv
└──┴──┘これはプリンターのテスト印字です。(EPSON VP-2500を使用).
```

〈リスト1〉プリンタ・ドライバ(MASM)

```
; IO.sys segment=430:
; ESC control included  1989 / 2 / 1

FIL     MACRO Q
        LOCAL Q1
Q1:     REPT  Q-Q1
        NOP
        ENDM
        ENDM

.SALL
kanm    EQU   3E24H
cnt     EQU   3E25H
prb     EQU   3E26H
pos     EQU   3E00H

CODE    SEGMENT
        ASSUME CS:CODE
START:  MOV     AX,430H
        MOV     ES,AX
        MOV     AX,CS
        MOV     DS,AX
        MOV     CX,287H-16BH
        MOV     DI,16BH
        MOV     SI,16BH
        CLD
P01:    MOVSB
        LOOP    P01
        INT     20H

        FIL     PMOJI
        ORG 06BH
P16B:
PMOJI:  MOV     CH,Byte Ptr DS:[prb]
        MOV     AL,CL
        CMP     CH,1BH ;ESC code?
        JAE     B16
        CMP     AL,1BH ;ESC code?
        JZ      P18B
        CMP     AL,80H
        JB      P190
        CMP     AL,0A0H
        JB      P18B
        CMP     AL,0E0H
        JB      P190
        CMP     AL,0FDH
        JNB     P190
P18B:   MOV     Byte Ptr DS:[prb],AL
        RET
P190:   JMP     ASCII

P193:
B16:    MOV     Byte Ptr DS:[prb],0
        JNZ     B17
        MOV     Byte Ptr DS:[kanm],0
        JMP SHORT ESCP
B17:    CALL    START+0F9CEH    ;50:38CEH

        ORG 0A2H
P1A2:
JIS:    PUSH    CX
        CALL    KANJ
        POP CX
        PUSH CX
        CMP     CH,2CH
        JNB     ZEN
        CMP     CH,29H
        JNB     HAN
ZEN:    CALL    SKIP
HAN:    POP     CX
        CALL    OT2B
SKIP:   MOV     CX,1B00H
        MOV     AL,Byte Ptr DS:[cnt]
        ADD     AL,7
P128:   INC     CX
        SUB     AL,3
        JNS     P128
        MOV     Byte Ptr DS:[cnt],AL
        DEC     CL
        JNZ     OT2B
        RET

SET320: MOV     CX,1C41H ; 3/20 mode
DEFAL:  MOV     Byte Ptr DS:[ELEET-START+1+pos],AL
        MOV     Byte Ptr DS:[SKIP-START+pos],0C3H
        MOV     Byte Ptr DS:[AS2-START+1+pos],B8A-AS2-2
        JMP SHORT OT2B

ESCP:   CMP     AL,48H
        JZ      PICA
        CMP     AL,45H
        JZ      ELIT
        CMP     AL,50H
        JZ      PROP
        CMP     AL,4EH
        JNZ     OT2B
        CALL    SET320
        MOV     Byte Ptr DS:[AS2-START+1+pos],B8B-AS2-2
        RET
PICA:   PUSH    CX
        MOV     CX,1C42H
        CALL    DEFAL
        JMP SHORT POPRT
ELIT:   PUSH    CX
        CALL    SET320
        MOV     Byte Ptr DS:[SKIP-START+pos],0B9H
        JMP SHORT POPRT
PROP:   PUSH    CX
        CALL    SET320
POPRT:  POP     CX
        FIL OT2B

        ORG 113H
P211:
OT2B:   XCHG    CH,CL
OT2C:   CALL    OT1CH
        MOV     CL,CH
        JMP SHORT OT1CH

        ORG 11CH
P21C:
ASCII:  PUSH    CX
        CMP     CL,13 ;cr code
        JNZ     AS2
        MOV     Byte Ptr DS:[cnt],0
AS2:    JMP SHORT B8A ; or B8B

ELEET:  MOV     CX,1B45H ;Eleet Char
        MOV     AL,Byte Ptr DS:[kanm]
        XOR     AL,1
        JZ      P115
PRET:   RET
KANJ:   MOV     CX,1B4BH ;Kanji yoko
        MOV     AL,Byte Ptr DS:[kanm]
        XOR     AL,1
        JZ      PRET
P115:   MOV     Byte Ptr DS:[kanm],AL
        JMP SHORT OT2B

        FIL     LIMIT
        ORG 152H
P252:
LIMIT:  RET

B8A:    CALL    ELEET
        POP     CX
        JMP SHORT OT1CH
B8B:    CALL    KANJ
        POP     CX
        MOV     CH,14H
        SHL     CX,1
        SHR     CL,1
        INC     CH
        CMP     CL,20H
        JNZ     P150
        MOV     CX,2B21H
P150:   JAE     OT2B

        FIL     OT1CH
        ORG 171H
P271:
OT1CH:  CMP     Byte Ptr DS:[27C1H],0FFH
        JZ      P280
P278:   CALL    START+5CH
        CMP     AH,1
        JNZ     P278
P280:   MOV     AH,11H
        MOV     AL,CL
        INT     1AH
        RET

CODE    ENDS
        END
```

出力コードをCLに入れて
このルーチンに入る

エンジニアへのプレテン・ボード ①

●君にも作れる高性能計算機アーキテクチャ●

計算機ハードウェア上の高性能化の方向は，次の三つに整理できます．

① IC化や実装など製造技術による高性能化
② 並列分散処理など構成方法による高性能化
③ 超伝導や光など新素材による高性能化

これらは単独に，またそれぞれ融合した形で研究&開発されています．①は今でも三次元構造化や信頼性などの技術課題が山積みしていますが，やや成熟技術の感を呈し，できて当たり前という見方をされています．

「話題」ということで歓迎を受けているのが②と③です．②の関係では，データ・フロー・マシン(その類似のコネクション・マシン)，ニューロ・コンピュータ，また個別技術としてASICやDSPによる一部機能の高性能化，あるいはRISC構成や多値，ファジィ論理構成などの方式工夫があげられます．どれもみな，ソフトウェア側からの要求によって工夫されたもの，と考えることができます．

③は光ファイバ，バイオ技術，高温超伝導で一躍脚光を浴びました．今のところはまだ先端技術(未開発技術と同義)領域でしょうか．物理など基礎科学分野の舞台になっています．

さて今回は，②の並列処理などの計算機アーキテクチャを掘り下げてみましょう．開発目標は常に速度/コスト比を向上させることにありますが，百人百様，さまざまなアーキテクチャが提起されています．

その中で広く知られているのがデータ・フロー計算機です．これは一言でいえば，完全ロジック回路と完全逐次処理ノイマン計算機の間の妥協の産物です．

ノイマン計算機(普通のマイコン)は，もともとは電子回路の集まりであり，ノイマン計算機で使用する命令は結果的には電子回路を動作させるものになります．したがって，これとは逆に，演算命令をランダム・ロジック，転送命令をレジスタICに，繰り返し命令をカウンタ回路で置き換えることにより，プログラム機能と等価な，しかもギリギリ高速なハード・ロジック回路を作ることも可能なはずです．現在のASIC(特定用途向けIC．例えば画像処理ICなど)の開発では，まさにこういう手順を踏むと聞きます．

計算機プログラムが原理的にはすべてハード回路に置き換えが可能であるとはいっても，回路規模が莫大になってしまいます．素子数や実装作業のコストを考えるなら，なんでもかんでもハード化するのは無理です．そこで妥協の産物が出てくるんですね．

マルチ・プロセッサの一形態であるデータ・フロー計算機がその例です．加算などのまとまった機能ブロックをノイマン型逐次処理構造で表現し，単一代入文という配線手法でそれらのブロック間をつなぎます．単一代入は機能ブロック同士の接続をひとつの変数(名)で行い，その変数をほかの箇所で使ってはならない，というルールです．変数を電線(結線ワイヤ)という言葉で置き換えましょう．

1本の電線は，ある機能ブロックの特定の端子に固定接続され，あっちこっちで共用してはいけない，というハード・ロジック結線と同じになります．もともと「データが流れるさまをイメージしながら設計する」というのは，ハード回路を作るときの設計手法です．ただし，データ・フロー計算機は結線や機能ブロック内容がプログラマブルであるところがハード・ロジックと異なります．

一方にノイマン計算機，他方にその極端な等価形態であるハード・ロジックがあり，それぞれ低速・低コスト，高速・高コストという関係があります．「計算機アーキテクチャ論」は，一般にノイマンからハード・ロジックの中間的妥協策として語られるもので，その機種たるや無尽蔵であると予測できます．評価の基準はハード価格であり，最も喝采を受けるのは，その時点でコスト・パフォーマンスの最適解を与えるアーキテクチャ，冒頭の①の条件にマッチした②の構成ということになりましょうか．いずれハード価格が低ければよりハード・ロジックに近づき，そうでなければノイマン型に近くなる，という開発傾向をもちます．

機能を固定したハード・ロジックの性格を考えると，マルチ・プロセッサの個数が多くなればそれだけ専用処理の方向に向かうはずです．「汎用高性能」マルチ・プロセッサなどは，汎用と高性能がそれぞれ相反の関係にあるので意味をなさないでしょうね．100個程度のマルチならまだノイマン的議論が通用するでしょうが，たとえば，100億個の「汎用」マルチ・プロセッサにいちいち制御プログラムを転送することは損か得か，ということを空想してみてください．

専用ハード化も極端に推し進めるとアナログ回路に向かうことも予想されます．これまで精度的に難点があったアナログ素子も，最近は見違えるような高性能になりコストも安くなってきました．代数演算の一部など，速い演算を望む場面ではアナログを採用したらどうでしょう．DSPの比ではないと思いますよ．

以上のように考えると，同じ機能表現をするといっても，ノイマンからハード・ロジック，あるいはアナログ回路までさまざまな方法と，それに付随してそれぞれ特徴あるプログラミング思想があることがわかります．②の問題はおおざっぱにはこの方法論を検討することになります．もちろんコストの問題もありますから①，③の技術進歩にも対処しなければなりません．それにしても，これからの計算機工学に携わる人は大変だなぁ…

8086アセンブラ・プログラミング入門 第1回

新連載

MS-DOS上でのマクロ・アセンブラの開発手順

8086
MS-DOS

相沢 一石

市販のワープロや表計算プログラムを利用しているだけなら，パーソナル・コンピュータはブラック・ボックスでもかまいません．それぞれパソコンは，ワープロ専用機や表計算専用機になっています．ところが，ブラック・ボックスでは困る場合があるのです．

例えば，自動車に乗るときでも普通に走るのであればボンネットを一度も開けずに自動車を乗りこなすことができます．しかし，ラリー・ドライバになると，運転操作だけでなく，それぞれの操作がどのような結果をもたらすか，車の特性についても知っていなければなりません．

すなわち，パーソナル・コンピュータにおいても，本来のコンピュータとして実験装置を制御したり，市販のプログラムで作成されたデータを自由に利用しようなどと考えると，そのためのプログラムを自分で作らなければなりません．

● アセンブラのプログラミングをとおしてコンピュータを理解してしまう

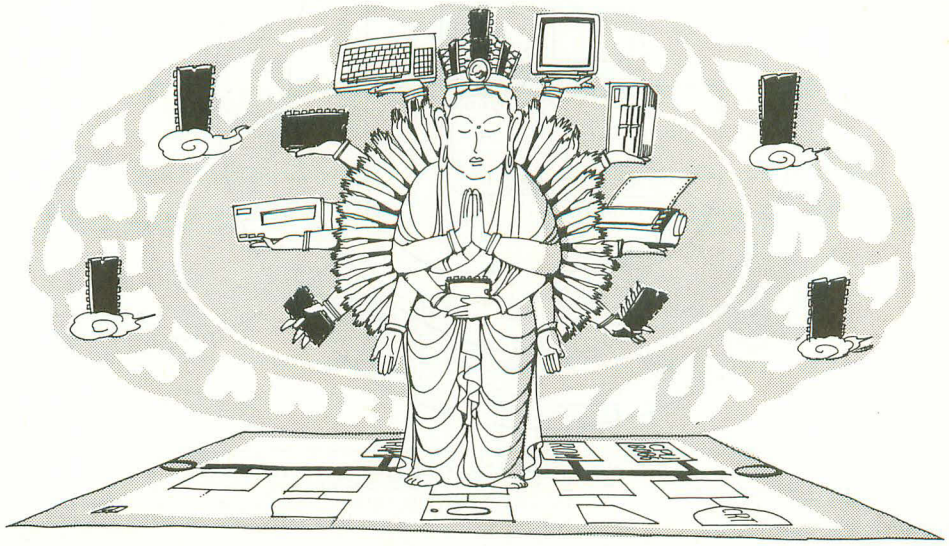
このプログラムを作ることは特別難しいことではなく，系統だって順番に理解していけば，だれでもひととおりのことができるようになります．また，アセンブラでプログラムを作ることで，コンピュータの仕組みまでわかってしまうようになります．

そこで，この連載ではパーソナル・コンピュータを最大限に利用するため，アセンブラのプログラムを具体的に作りながら，プログラミングの方法を説明していきます．例えば，通信の処理やパーソナル・コンピュータに新たな入出力装置を追加した場合など，それらの処理のためにアセンブラでプログラムをつくります．そして，まったくの初心者でもPC98シリーズのパーソナル・コンピュータを用いて，アセンブラのプログラミングの作成ができるようにします．

最初は簡単なものから始めて，最終的にはフロッピ・ディスクの読み書きを行うプログラムを作り，さらに誤って削除してしまったファイルを復活するなどという アセンブラのプログラムでなければできないユーティリティも作ります．

したがって，この連載を体験することにより，たんに計算処理のプログラムでなく ハードウェア制御プログラムを作成する実力を養成すると共に，より深くパーソナル・コンピュータの仕組みを理解することができるようになります．

〈図1-1〉マイクロコンピュータ・システム



連載の予定

① パーソナル・コンピュータでプログラムを作るための環境の整備
　▶一番簡単なプログラムを具体的に作る
　▶アセンブルの方法を示す
② マクロ命令を用いたプログラムの作り方
　▶メモリ・ダンプのプログラムを作る
③ ライブラリの使い方
　▶ FDC を制御するプログラムを作る
④ MS-DOS の機能を利用したディスクの読み書きルーチンの処理
⑤ BIOS を利用した任意のフォーマットのディスクの処理プログラムの作成
⑥ ディスク・エディタの作成

以上のような内容を予定しています．したがって，全体で 6 から 10 回になる予定です．

アセンブラでプログラムを作るための環境

アセンブラを使う前に，現在一番使われているパーソナル・コンピュータ用の OS(オペレーティング・システム)である MS-DOS について簡単に説明します．

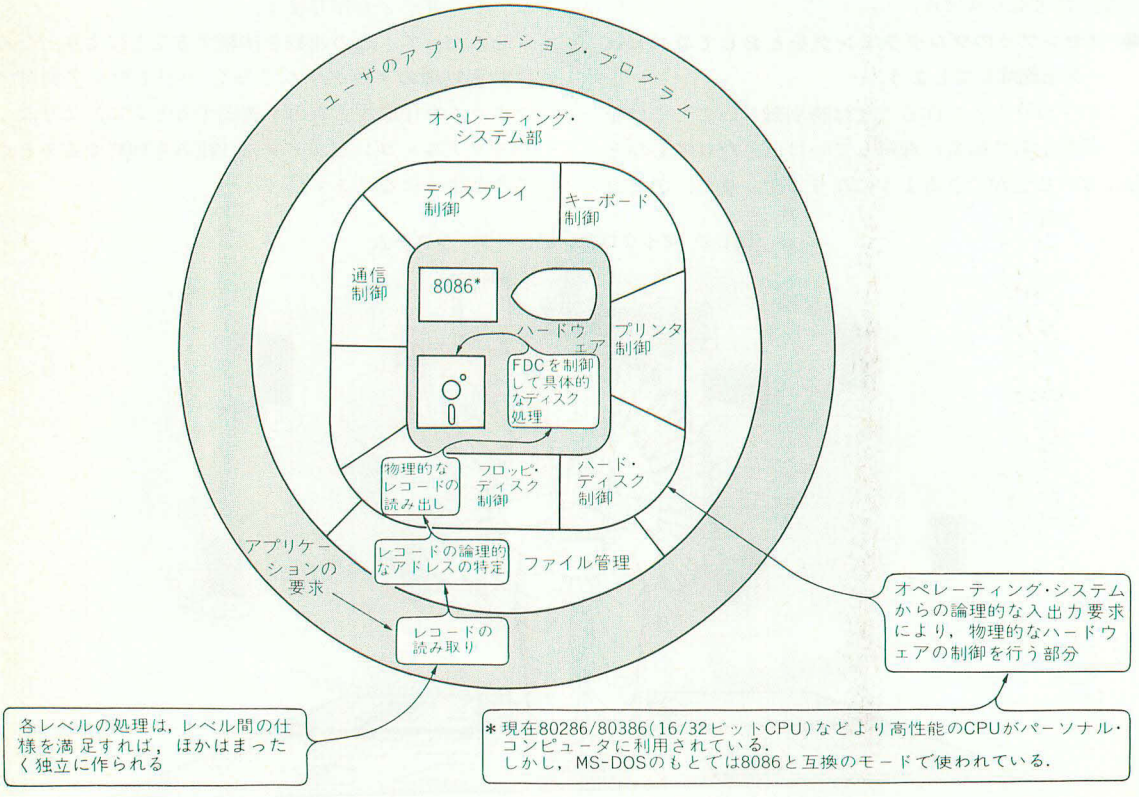
● ハードウェアの制御をするのは OS の中の BIOS や IOSYS

コンピュータ・システムは図1-1 に示すようなハードウェアの構成になっています．このハードウェアを制御するソフトウェアは，一般的なパーソナル・コンピュータでは図 1-2 に示すような階層構造になっています．具体的なハードウェアの制御を行うのは BIOS (バイオス)，または IOSYS(アイオーシス)などと呼ばれる部分です．OS では，ここだけをハードウェアに依存する部分としてプログラムが作られています．

そして，オペレーティング・システムの本体部分では，直接ハードウェアを操作することはありません．必ず IOSYS などのハードウェアの制御ルーチンを利用して，入出力などのハードウェアの処理を行います．そのようにすることで，ハードウェアが異なったとしても，IOSYS への処理の要求が同一になり，ハードウェアの差は IOSYS の中で吸収されてしまいます．

このような仕組みになっているので，メーカの違うパーソナル・コンピュータであっても，同じ OS 対応のプログラムであれば，そのまま利用することができます．しかし，ソフトウェアによっては，とくに画面処理の部分で独自にハードの処理を付加したり，IOSYS を経由するというルールに従わないで，プログラムの互換性を損っているものもありますが，一般的には標

〈図1-2〉 オペレーティング・システムの階層構造



準化を進める努力が払われています.

オペレーティング・システム(OS)がもつ機能

オペレーティング・システムは，そのシステムを利用する人のために，各種の援助プログラムを用意しています．図1-3に，MS-DOSの場合のオペレーティング・システムで用意されているものを示します．

① ファイルの管理を行うためのプログラム
- ▶ファイル名の一覧表の表示
- ▶ファイルの削除，複写
- ▶ファイルの内容の表示

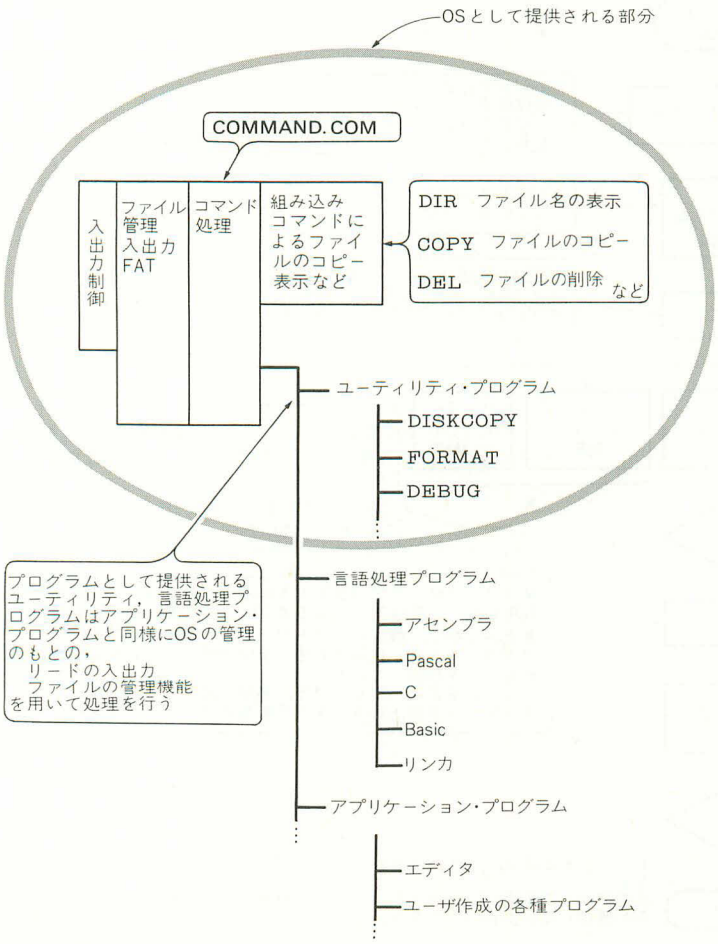
② プログラムの実行管理を行うためのプログラム
アプリケーション・プログラムをOSの管理の下で実行する

③ 各ユーティリティ・プログラムでプログラムの開発，運用に役立つプログラム
- ▶エディタ……ソース・プログラムをコンソールの画面とキーボードを使って作成するためのプログラム
- ▶アセンブラ…アセンブリ言語で書かれたソース・プログラムをコンピュータが実行可能なコードに変換する
- ▶コンパイラ…Cobol，Fortran，Pascal，Cなど高級言語と呼ばれる言語で書かれたソース・プログラムを翻訳する
- ▶リンカ……各言語で記述されアセンブラ，コンパイラで変換，翻訳された複数のプログラムを結び付け，それぞれの機能を利用した単一の実行可能なプログラムにする

また，オペレーティング・システム上のユーティリティは，新しい要求に対してそれぞれプログラムが作られるので，OSがバージョン・アップされるたびに強化されています．

さらに，利用者が独自にユーティリティを作ることもできます．高級言語を使用しているときには，独自にユーティリティ・プログラムを作る必要性はそれほど感じないのですが，アセンブラを使用しているときは，入出力ルーチンの記述でさえもプログラマが作成しなければなりません．

〈図1-3〉 MS-DOS オペレーティング・システム



ただし，システムに接続される各入力装置との基本的な処理ルーチンはOSによって用意されています．例えば，コンソールなどとの1文字ずつの入出力を行うものから，ディスク・ファイルのオープン/クローズ，レコードの入出力，メモリの管理などの基本的な処理はひととおり用意されています．表1-1に，MS-DOSが用意しているキャラクタの入出力のファンクション機能を示しておきます．

また，MS-DOSでは，新たに入出力装置をシステムに追加した場合でも，この入出力装置を標準のOSの管理下で制御するための標準の入出力プログラムを付加することができます．これら新規に追加する入出力プログラムのことをデバイス・ドライバと呼び，このデバイス・ドライバを作成するための詳細な仕様が公開されています．後ほど具体的なデバイス・ドライバを作りながら説明する予定です．

プログラムの開発手順

MS-DOSのもとでプログラムを作成する具体的な手順を示すと図

1-4のようになります．

ソース・プログラムを作成するためのエディタには様々なソフトウェアが市販されていますし，MS-DOSのテキスト・ファイルを作成できるワープロをソース・プログラムの作成に利用することもできます．

したがって，読者の方は，一番使い慣れたエディタまたはワープロを用いて，ソース・プログラムを作成してください．

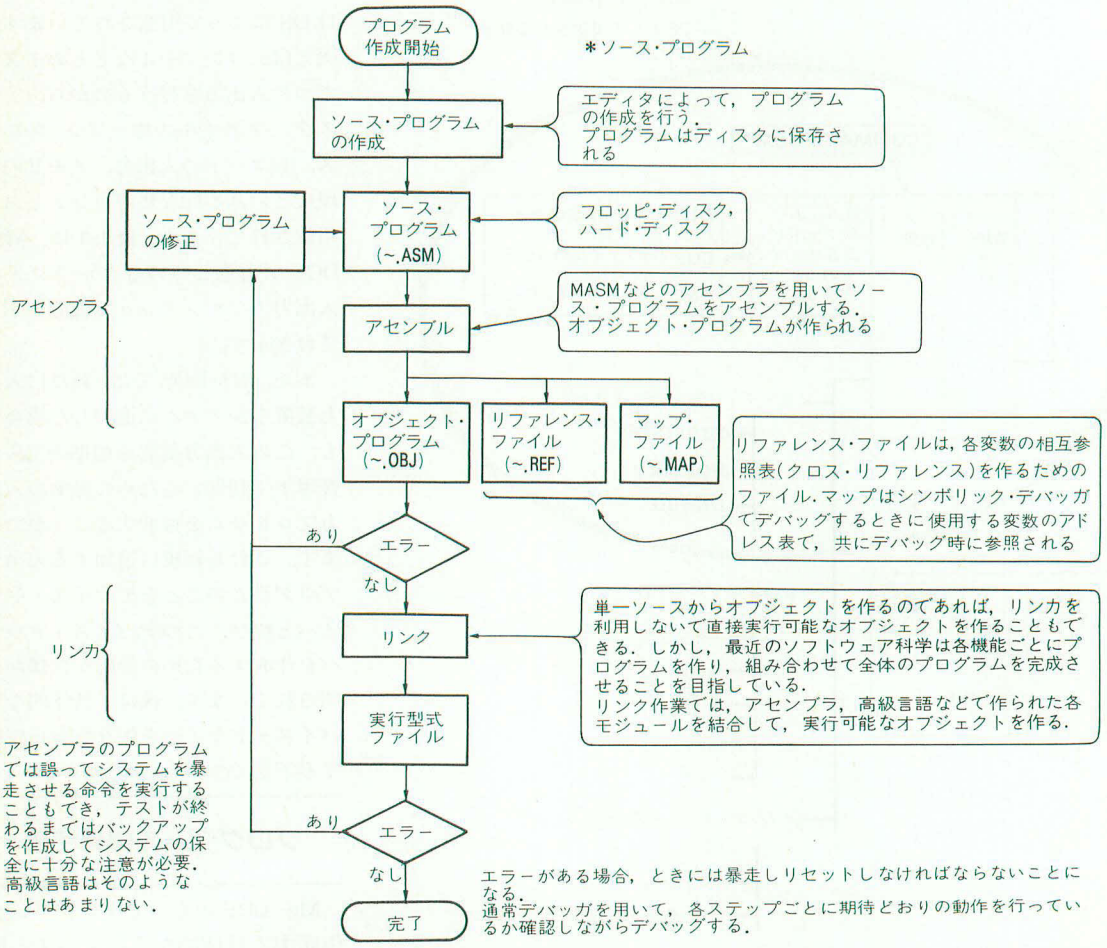
初めてプログラムを作成する場合はソース・プログラムの記述方法がわからないかもしれません．また，

〈表1-1〉キャラクタ・デバイス入出力のファンクション機能

コード	機　　能	説　　明
01H	キーボード入力とエコー	コンソール入力から1文字受け取ると，それをコンソールに出力する
02H	文字のスクリーン出力	コンソールに1文字出力する
03H	補助入力	補助入力装置(RS-232Cインターフェースなど)から1文字受け取る
04H	補助出力	補助出力装置に1文字出力する
05H	文字のプリンタ出力	プリンタに1文字出力する
06H	直接コンソールI/O	コンソール入力から1文字受け取るか，コンソールに1文字出力する
07H	直接コンソール入力	コンソール入力から1文字受け取る
08H	キーボード入力	コンソール入力から1文字受け取る
09H	ストリングのスクリーン出力	コンソールに文字列を出力する
0AH	バッファード・キーボード入力	コンソール入力から文字列を受け取る
0BH	キーボード・ステータスの検査	コンソール入力のバッファのステータスを返す
0CH	バッファを空にして，キーボード入力	コンソール入力のバッファを空にしてコンソール入力から1文字受け取る

└MS-DOSで提供される機能を示すコード．この値をAHレジスタにセットし，**INT 21H**の内部割り込み処理で機能の実現をDOSに託す

〈図1-4〉プログラム作成の手順



今回説明するアセンブラのプログラムの場合は高級言語と異なって，表示方法だけをとってみても2進数，10進数，あるいは16進数のどれにするかなど，データ処理の細部についても具体的にプログラミングしなければなりません．さらに，データの処理以外にもCPUのもつ制限，または具体的なデータ処理のハードウェアの制限，処理方法についても考慮を払わなければなりません．その点が，アセンブラでプログラムを作ることを難しくしています．

それにくらべると高級言語では，利用者のイメージを実現する方法を考えればよいのです．高級言語は，利用者が具体的にイメージしやすいように言語体系が作られているからです．アセンブラでは，利用者のイメージをCPUの機能で実現するためのハードウェア，OSの機能で実現するための処理の組み立ても必要です．

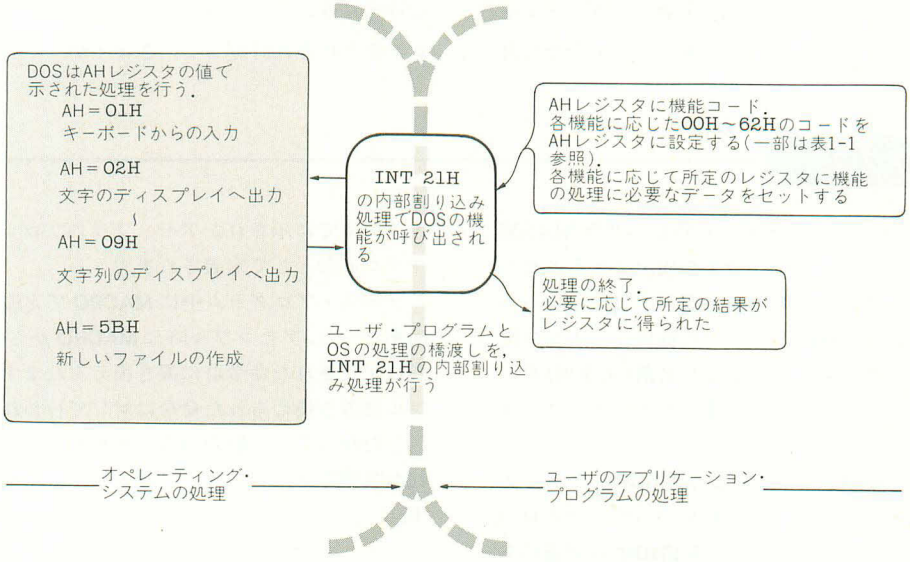
それぞれのハードウェアやOSでは，プログラマのためにプログラミングに必要な基本的な機能をあらかじめ用意しています．MS-DOSの場合は，DOSの機能として**図1-5**に示すような機能が用意されています(**表1-1**参照)．

MS-DOSでのアセンブラのソース・プログラムの作成方法

アセンブラは，基本的にはインテル社のマクロアセンブラの記述を踏襲していますが，インテル社のマクロアセンブラも一つの開発システムであることにかわりはなく，それぞれのアセンブラを開発したメーカごとに，使いやすくするための工夫や機能が追加されていることがあります．

ここでは，マイクロソフト社のMASMというマクロ・アセンブラの記述方法を中心に説明します．

〈図1-5〉MS-DOSで提供される機能の利用法



コラム　ファイル型式

MS-DOSの管理下では，プログラムは次の二つのファイル型式となります．

COM型式……コード，データ，スタックの各セグメントが同一の物理セグメントに配置されコンパクトなプログラムです．また，オブジェクトは次のEXE型式と異なり，メモリへのロード時にDOSによるリロケータブルな変数のアドレス変換などの作業を必要としません．

したがって，デバイス・ドライバ，高級言語からアセンブラのサブルーチンとして利用されるモジュールなどはこの型式です．

EXE型式……MS-DOSでもっともよく使われる8086の本来のプログラム型式で，データ，命令部，スタックの各セグメントが独立に設定でき，大きなプログラムが作成できます．

通常のプログラムでは各セグメントが64Kバイト以上になることはあまりありませんので，セグメントの大きさをとくに考慮することなくプログラムすることができます．

アセンブラの説明ではCPUの機能を考慮することも不可欠ですので，合わせて8086のハンドブックなどを参考にしてください．

それでは，具体的なプログラムを作ってみます．最初はできるだけ簡単な例から始めましょう．ここで作成するサンプル・プログラムは次のような処理を行います．

① 画面に次のようなメッセージを出し，プログラムが開始したことを示します．

「今日から，毎日勉強します．
次に進む時はスペースキーを押して下さい」

② キーが押されるのを待ち，スペース・キーが押されたら次に進み，それ以外なら最初の表示を繰り返します．

③ DOSの制御に戻ります．

この処理を具体的にプログラミングしたものが**リスト1-1**です．

アセンブラのプログラムには命令コードのニモニックだけではなく，アセンブラのシステムにソース・プログラムをアセンブルするための条件を示す命令も書かれています．

アセンブルするということは，ニモニックなどのアセンブリ言語で書かれたソース・プログラムを機械語コードに変換する一連の操作をいいます．ただし，リロケータブル・アセンブラであるMASMでは，実行可能なプログラムにするまでには種々の操作が必要で，これらの操作も含めて，アセンブルするといいます．

このアセンブル作業を制御するための命令を疑似命令といい，**リスト1-1**で使用した命令の機能を**図1-6**，**図1-7**に示します．

● SEGMENT, ASSUME 疑似命令

SEGMENT疑似命令はソース・プログラムで使用するセグメントを設定し，セグメント・レジスタの名前やつなぎかた(アライン)などを指定します．各セグメントの設定には必ず，**SEGMENT**(始め)，**ENDS**(終わり)の二つの疑似命令を使います．

ASSUME疑似命令は，各セグメント・レジスタの値を仮に決めるものです．そして，アセンブル時にアセンブラがソース・プログラムのデータやラベルのアドレスをそれぞれ計算していきますが，そのときカウン

コラム　マクロ命令

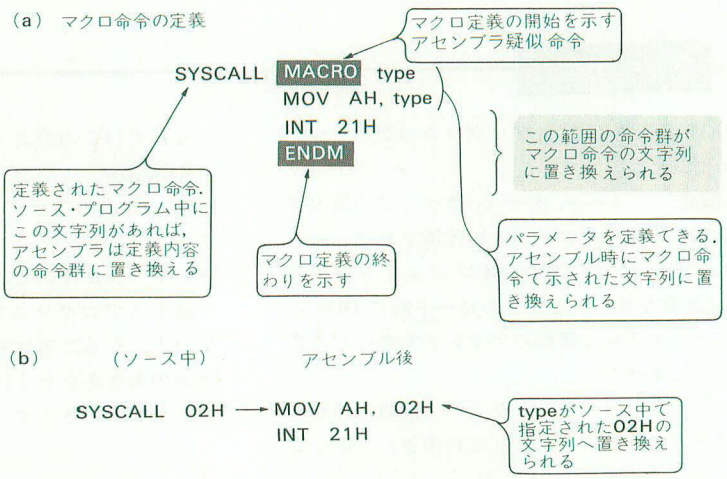
マイクロソフト社のマクロ・アセンブラMASMは，名の示すとおりマクロ命令を使用できるアセンブラです．マクロ命令の機能は繰り返し使用する一連の命令，特定の機能をもたせた処理ルーチンの代わりに，ユーザが新たに設定した名前(文字列)を，あたかも新しい命令のように使用できるようにします．

具体的には図1-Aに示すよう**MACRO**というアセンブラ疑似命令で，マクロ命令の名前とマクロの定義の開始を示します．これ以後**ENDM**の記述のある部分までに示されたアセンブラのプログラムが，マクロ命令として定義されます．

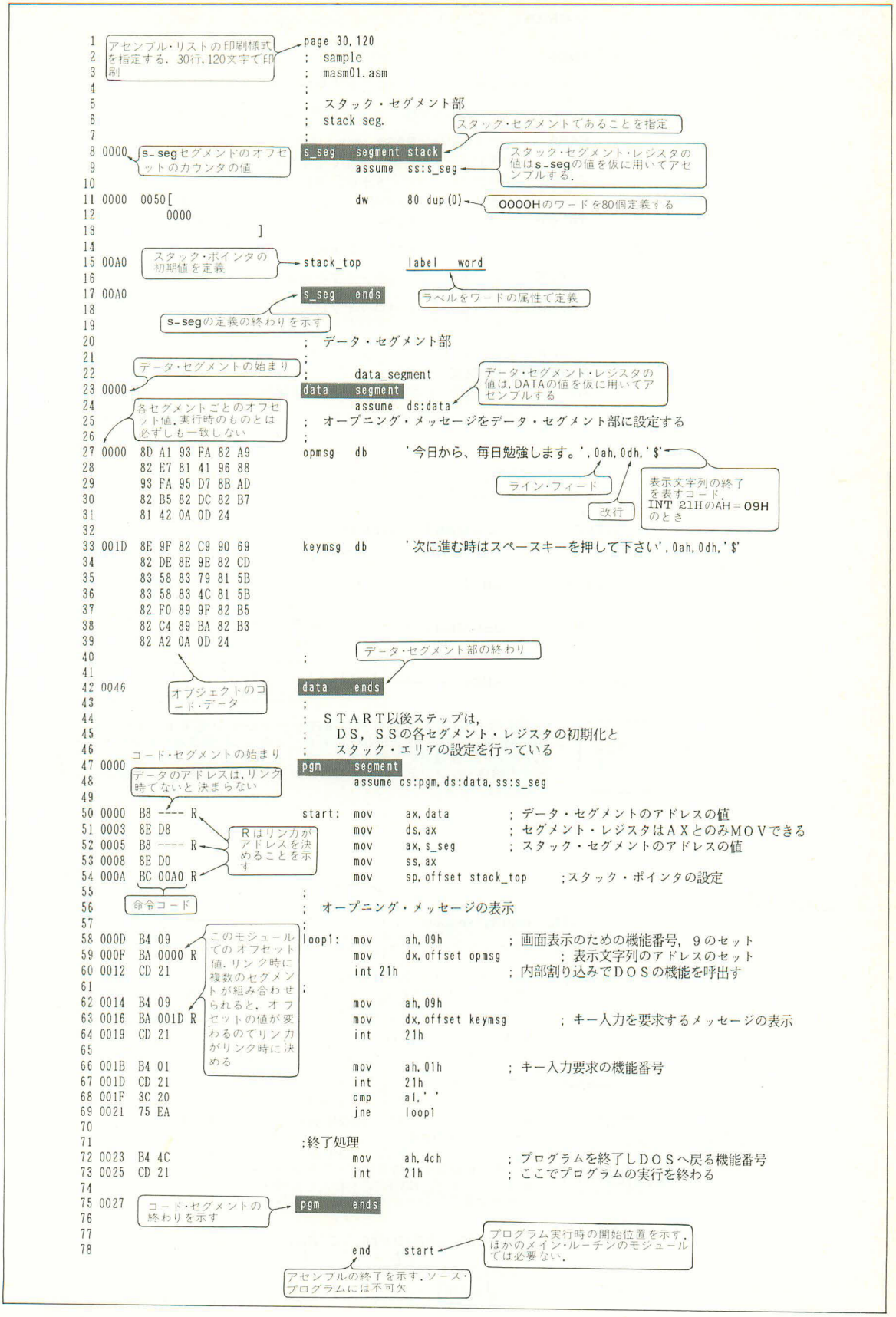
ソース・プログラム中に**MACRO**で定義された名前を書くと，アセンブル時に**MACRO**から**ENDM**の間に記述された命令群に置き換えられます．アセンブルは置き換えられた命令に対して行われます．

したがって，一群の命令の代わりにマクロ定義された名前でソース・プログラムを作ることができます．

〈図1-A〉
マクロ命令の定義と使い方



〈リスト1-1〉メッセージを出すサンプル・プログラム



〈図1-6〉
アセンブラ疑似命令 SEGMENT/ENDS

SEGMT/ENDS疑似命令

セグメント名　SEGMENT　〔アライン・タイプ〕〔 結合タイプ〕〔 'クラス名'〕
　　　⁞
セグメント名　ENDS

セグメント名	セグメントの名前　アセンブラの名前の条件に従う	
アライン・タイプ	セグメントのロードされる物理アドレスの境界の条件を示す．	
	PARA	デフォルト値はPARAになる．境界アドレスは16の倍数になる
	BYTE	境界アドレスはバイト単位で任意のアドレスから始まる
	WORD	境界アドレスはワード単位，偶数アドレスから始まる
	PAGE	境界アドレスは256の倍数の任意のアドレスから始まる
	INPAGE	セグメント全体が256バイト以下でページをまたがらない
結合タイプ	デフォルト	同一モジュール内の同じ名前のセグメントとのみ組み合わせる
	PUBLIC	同じ名前でこの結合タイプが指定されたセグメントは一つの物理セグメントに結合される
	COMMON	同じ名前でこの結合タイプが指定されたセグメントは同じ物理アドレスに重ね合わせて一つの物理アドレスとする．セグメントは最大のセグメント長となる
	STACK	このセグメントはスタック・セグメントであることを示す
	MEMORY	PUBLICと同等
	AT	ATに対してはアドレスを示し，直接物理セグメントを指定することができる
クラス名	クラス名の指定によって同じクラス名をもった論理セグメントがまとめて配置される	

【例】

```
(1) STACK    SEGMENT  STACK
                ⁞
    STACK    ENDS

(2) CODE     SEGMENT
                ⁞
    CODE     ENDS
```

（注）セグメント名の指定は不可欠

（注）スタック・セグメントにはSTACKのコンバイン・タイプが不可欠

（注）通常スタック・セグメント以外ではアラインメントなどはデフォルト値でよい

（注）ソース中にほかの場所に同一の名前のセグメントの記述もできる

（注）ソース中のほかの場所に同一の名前のセグメントの記述もできる

SEGMENT 文は以下の文がどの論理セグメントに属するかを指定する．
SEGMENTには対応するENDSがあり，この二つに囲まれた部分がセグメント名で示された論理セグメント名となる．
同一の論理セグメント名がある場合，リンカは同一の連続したセグメントとして処理する

〈図1-7〉
ASSUME 疑似命令

ASSUME　segreg : segpart 〔, …〕

（注）これはカンマで区別しながらどんどん書いていってもよいし，省略してもよいということ

アセンブラにアセンブル時のセグメント・レジスタの値をどのように設定するか指定する．各セグメント・レジスタにセグメント名を対応付ける．

segreg　各セグメント・レジスタを示す．
- CSは，コード・セグメント・レジスタを示す（注：プログラムもしくはプログラム＋データが入るところ）
- DSは，データ・セグメント・レジスタを示す
- SSは，スタック・セグメント・レジスタを示す
- ESは，エキストラ・セグメント・レジスタを示す

segpart　対象となるセグメント名を示す．
セグメント名として次のものが使用できる．
- ▶ソース・ファイル中で，SEGMENT疑似命令で定義されたグループの名前
- ▶ソース・ファイル中で，GROUP疑似命令で定義され以前のレジスタの指定はすべて取消される
- ▶キーワードで，NOTHINGを指定した場合，それ以前のレジスタの指定はすべて取消される

（注）ASSUME 命令は，アセンブラに対して，各セグメント・レジスタに具体的なセグメント名を対応づけるだけで，各セグメント・レジスタの値はMOV命令などでプログラム中で記述することになる

```
assume  CS:CODE, DS:DATA, SS:SS1
```

CODE：コード・セグメントの論理セグメント名
DATA：データ・セグメントの論理セグメント名
SS1：スタック・セグメントの論理セグメント名

論理セグメント名はセグメント疑似命令で指定する

タとして使用するセグメントの名前を指定します.

● コード・セグメントの最初には各セグメント・レジスタの値の初期化が必要

マクロ・アセンブラで開発されたプログラムを実行するのは,通常 MS-DOS の制御のもとで行います.この場合,MS-DOS はディスクなどのプログラムの格納場所からアセンブラによるプログラムを読み出してきて,メモリへ配置する作業を行ってから,制御をそのプログラムに移します.

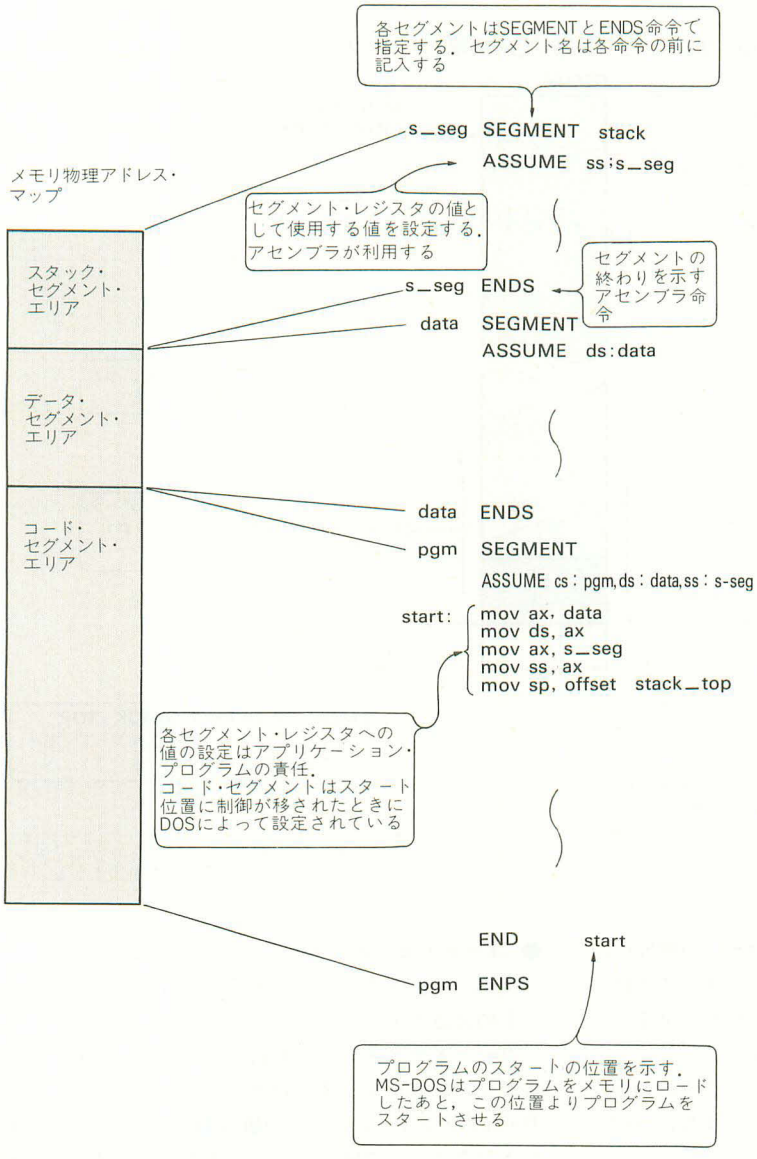
このとき,コード・セグメントの値は初期化されますが,ほかのレジスタの値は MS-DOS の活動状態のままになっています.そこで,ASSUME 疑似命令で指定したセグメント・レジスタの値によって,実際の各レジスタの初期化を行います(図 1-7 参照).

この初期化は,DOS の制御とアプリケーション・プログラムとの制御の受け渡しの間で生じることで,メイン・プログラムから呼び出されるモジュールはそのモジュールの設計にしたがった処理を行います.メイン・プログラムでのスタックやデータ・セグメントの値を使用するのが普通ですので,セグメント・レジスタの値を意識することなくプログラムすることができます(図1-8 参照).

ソース・プログラムは図 1-9 を参照して作成してください.次に,作成したソース・プログラムの説明をします.

サンプルのソース・プログラムの説明

〈図1-8〉EXE 型式のオブジェクトのメモリ・マップ



リスト1-1 に示したサンプル・プログラムでは,スタック,データ,コードの順に各セグメントの定義を行っています.以下に各セグメントの説明をしますが,実際のプログラムの動作はコード・セグメントに書かれています.したがって,サンプル・プログラムの動作についてはコード・セグメントの説明の中で行っています.

● 各セグメントの定義

最初にスタック・セグメントの設定を行います.

スタック領域を 80 ワード,DW 命令を使って設定します.同じ命令を 80 回続けて記述することもできますが,ここでは DUP 命令で 80 ワードの設定を一行ですませています.

● スタック領域とは

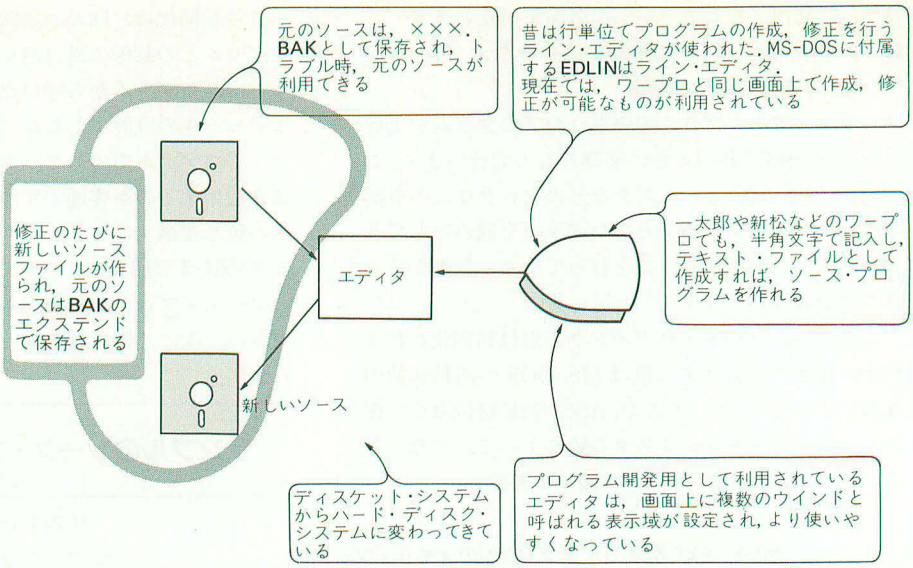
スタック領域は図1-10 に示したように,サブルーチンを呼び出したときの戻り番地を記憶したり,このサブルーチンと呼び出し元のプログラムとの間でのデータの受け渡しを行うときなどに利用されます.

● データ・セグメント

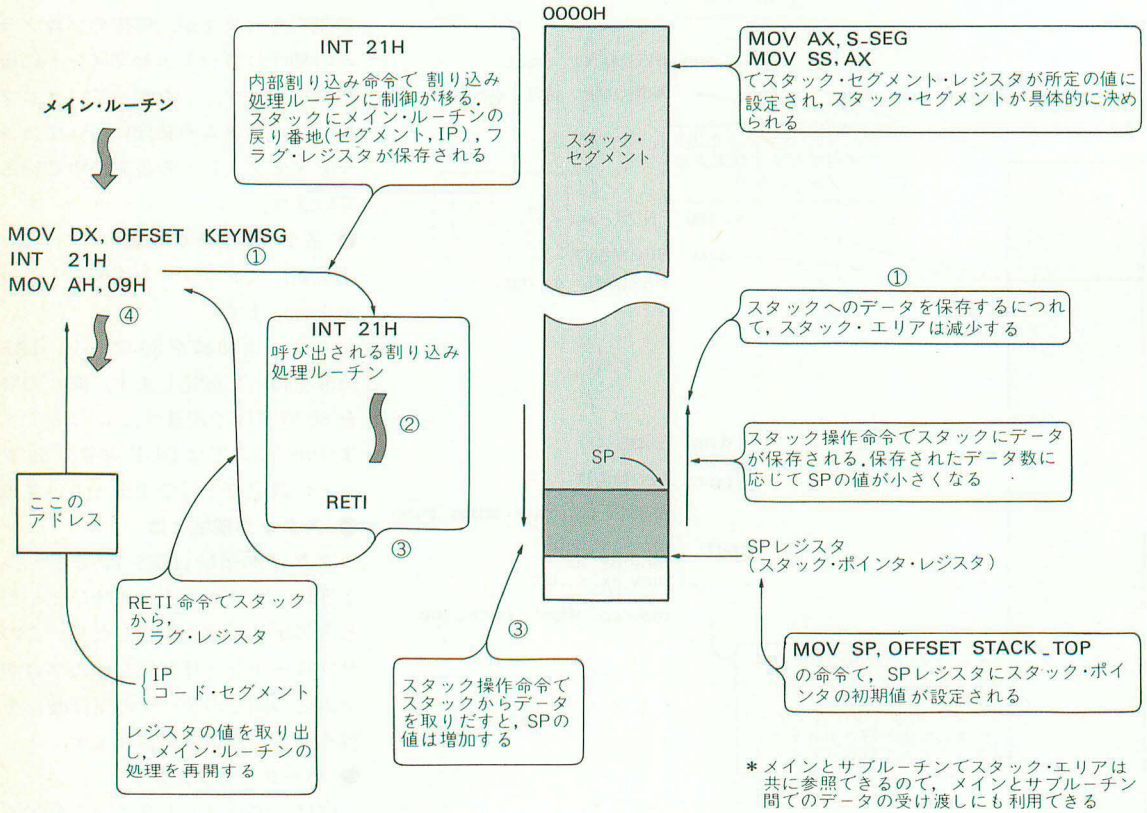
次に,データ・セグメントを記述します.この場合,データ・セグメントでは,画面に表示するデータを用意します.

文字データはバイト・データとして設定しますので,DB 命令を使用します.バイト・データの設定であっても,一つの命令の定義で連続したデータの定義ができます.最後の

〈図1-9〉
エディタはソース・プログラムの作成，修正を担当する



〈図1-10〉サブルーチン，割り込み処理とスタック



$マークは，画面にデータを表示するDOSの機能に，表示データの終了を知らせるためのものです．この終了マークを忘れると，画面一杯にあらぬ模様，文字が表示され故障かと慌てることになります．

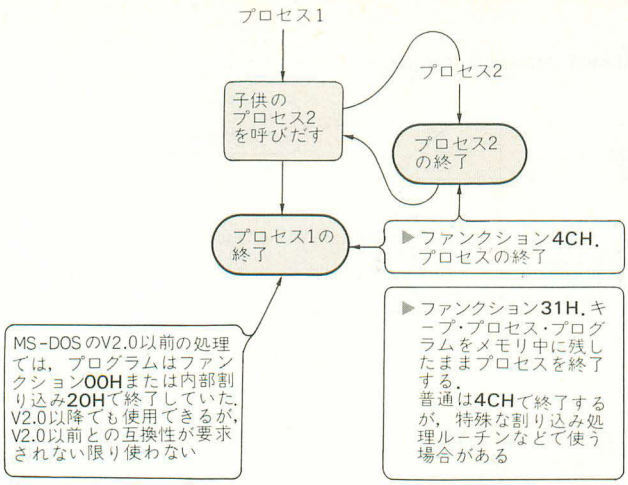
データの定義の前にあるラベルは，プログラムの中でデータを参照するときに使用されます．また，ラベルはデータの名前になります．

● コード・セグメント

次に，プログラムの実行部分であるコード・セグメントの記述です．

コード・セグメントの最初に，プログラム実行時の各セグメント・レジスタの値の設定を行います．コード・セグメント・レジスタの値はDOSによって実行すべきセグメントの値になっていますが，スタック，デ

〈図1-11〉プログラムの実行単位プロセスの終了



MS-DOSではプログラムの実行単位をプロセスと呼ぶ.プロセスは実行途中に自分自身の処理を中断しほかのプロセスを実行することができる.このとき,実行されるプログラムはそれぞれ独立なものであっても処理の手順として親子関係が生じ,子供のプロセスが終了すると順次親のプロセスに処理が戻っていく.

＊MS-DOSもV2.0以前と以後では大きく変わっているので,とくに必要ない限りV2.0以前のみの機能は使用しないようにする

ータ，エキストラの各セグメントの値は，DOSの使用していた値のままになっています．そのためこのプログラムの実行環境に適した値にしなければなりません．

それぞれ，各セグメントで仮定した値ですのでAXレジスタ経由で設定します．これにはMOV命令を使用します．データの移動にはこのMOV命令が大活躍します．またアドレッシングによっても，いろいろなデータを指定することができます．

ここでは，DS, SSの各セグメント・レジスタとスタック・ポインタの設定を行っています．

● 始めの挨拶の表示

オープニング・メッセージの表示は，DOSの文字列表示の機能を利用します．AHレジスタにどの機能を使用するのかを示すコード，この場合**09H**をセットします．それぞれの機能に応じて所定のレジスタに必要な値をセットします．ここでは文字列の先頭アドレスのオフセットの値をDXレジスタにセットしています．

● キーボードからの入力

キー入力の機能コードは**01H**で，入力データがあるとALレジスタにその値がセットされ戻されます．

キー入力データのチェックはCMP命令で行います．比較した結果が各フラグにセットされ，次の条件付きのジャンプ命令でそれぞれ該当する場所へ制御が移されます．

● プログラムの終了

プログラムの終了処理もDOSの機能を利用しています．すなわち，DOSの制御に戻る機能コード**4CH**をセットしています．

プログラムの終了といっても，今のプログラム処理のあとに何をするかによって，いくつかの終わり方があります．後に何もすることがなければ，“電源を切ってください”と表示してHLT命令を実行するだけですみます．

しかし，普通はプログラムの実行が終わるとDOSの制御に戻り 次の処理を行えるようにします．高級言語では，プログラマはとくに意識することなくこれらの処理をコンパイラが処理してくれます．アセンブラでは，DOSの機能で用意されている終了方法から選ぶことになります．これら終了方法については図1-11に示します．

● END命令

プログラムの最後には，END命令でアセンブルの終了を指示します．END命令の後のラベル名はプログラムの開始場所を示します．

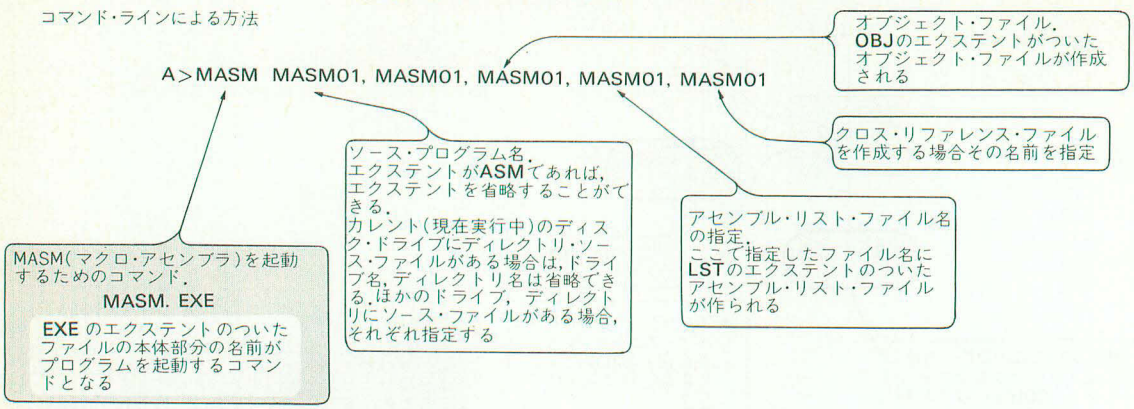
このEND命令はプログラムの終了を示すのではなく，アセンブル作業の終了を示すためソース・プログラムの最後に書いてください．誤ってプログラムの途中にこの命令を入れると，以後のソースはアセンブルの対象とならず中途半端なオブジェクトとなってしまいます．当然，多くのエラーが表示され戸惑うことになります．

アセンブル作業

ソース・プログラムの作成が終わると，MASMを用いてアセンブルを行います．アセンブル時にエラーがあると画面にエラーの内容，場所が表示されます．エラーの数が多くて画面に入りきらない場合，始めの部分がスクロールされて見えなくなります．

画面をプリンタにモニタするモードでアセンブルするとエラーの状況が印刷されます．これはCTRLキーを押しながらPのキーを押すことで，プリンタへの印字のON/OFFが交互に切り替えられます．

〈図1-12(a)〉 MASM によるアセンブル方法



〈図1-12(b)〉 アセンブル操作

```
パラメータを指定しないと，各四つのファイルの名前の入力を要求してくる
A:>masm
Microsoft  (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

Source filename [ .ASM]: masm01 ↵ ソース・ファイル名を入力．エクステントはデフォルトでASM
Object  filename [masm01. OBJ]: ↵ オブジェクト・ファイル名は，デフォルトでソース・ファイル名と同じになる
Source listing  [NUL.LST]: masm01 ↵ リスト・ファイル名の入力
Cross-reference [NUL.CRF]: masm01 ↵ クロス・リファレンス・ファイル名の入力
                                                デフォルトはNULでファイルを作成しない

  47004+269824 Bytes  symbol  space  free

       0 Warning Errors
       0 Severe  Errors      エラーの内容，合計を表示しアセンブラを終わる

パラメータを指定すると，ファイル名の入力要求なしでアセンブルを始める
A:>masm masm01, masm01, masm01, masm01, masm01
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

  47002+269746 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors
```

ソース・ファイル名だけは必要であるが，以後のファイル名はデフォルト値でよい場合；を示しておけば，ファイル名の入力要求は抑止される．
LINKも同様

また，アセンブル時にアセンブル・リストの出力を指定すると，アセンブル・リストがディスクに作成されます．アセンブル・リストにはエラーの情報がありますので，その情報にしたがってソース・プログラムの修正を行います．

そして，エラーがなくなるまでアセンブル，修正を繰り返します．

リンク作業

MASM によるアセンブル作業で得られたオブジェクト・プログラムは，まだそのままでは実行できません．LINK(リンカ)によってライブラリなどとのリンクを行って，実行可能なプログラム(EXE ファイル)が作られます．

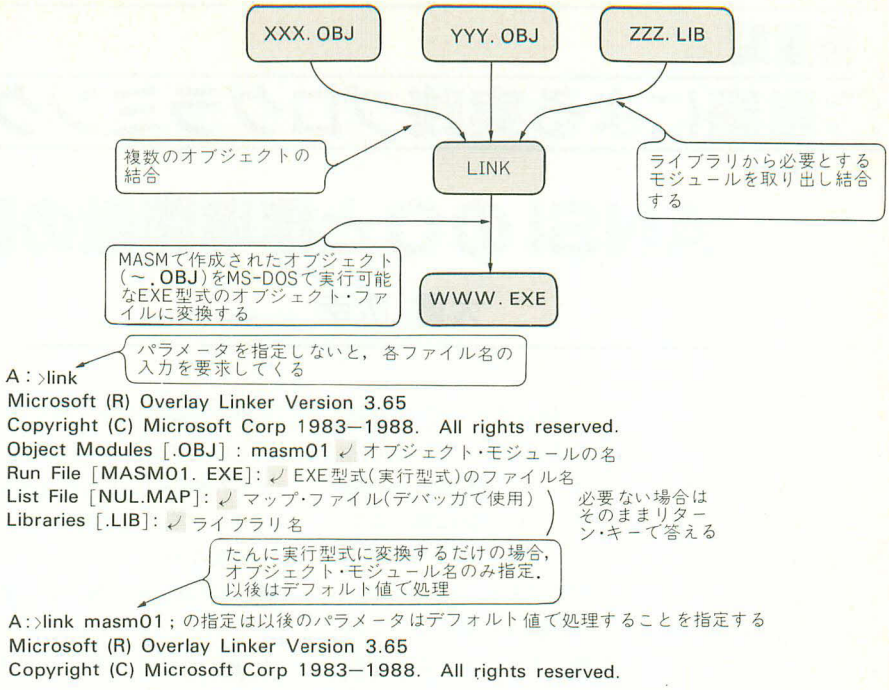
今回は，アセンブルされたプログラム単独で実行しますので，ほかのオブジェクト・モジュールとのリンク(結合)はありません．しかしリンカによって，アセンブラで作成されたオブジェクトから実行可能な EXE 型式のオブジェクト・プログラムを作成します．

リンカはスタックの有無，ラベル，データの参照の矛盾などのチェックを行い，エラーの表示も行います．

次回は，今回作成したプログラムをデバッガを用いてデバッグする方法を説明します．そのあと，マクロ命令を用いたプログラムの例を考えます．また，メモリの内容を表示するための方法も検討します．メモリの内容はバイナリのデータとして保存されていますから，確認のためには画面やプリンタへ文字として表示しなければ利用者には認識できません．これらの変換の方法なども，具体的な例をプログラムする予定です．

今回でてきた，アセンブラの命令の一覧を表 1-2 に示します．

〈図1-13〉
リンクの操作



〈表1-2〉今回でてきたアセンブラ命令

命　令	使　い　方	機　能
疑似命令		
segment	s_seg segment stack	segment の開始および属性を設定する．
assume	assume ds:data	セグメント・レジスタの論理的な値を仮定する．
ends	s_seg ends	同一の名前をもった segment 命令で開始されたセグメントの記述の終了を示す．
dw	dw 80 dup(0)	ワード・データの指定．
db	opmsg db	バイト・データの指定．
end	end start	アセンブルの終了を示す．名前が付いている場合はプログラムの実行開始場所を示す．
dup	10 dup(0)	繰り返し．この場合 0 を 10 回．
ニモニック		
mov	mov ds,ax	ソース・オペランドの内容をデスティネーション・オペランドにコピーする．ソースオペランドの内容は変わらない．
int	int 21H	内部割り込みを示す．数字は割り込みベクトルで 00～FFH まで
cmp	cmp AL," "	デスティネーション・オペランドからソース・オペランドを引き，その結果によりフラグがセットされる．オペランドの内容は両方共変わらない．
jne	jne loop1	等しくないときジャンプする．

Keyword

BIOS　：Basic Input Output System の略で直接ハードウェアの制御を行い基本的な入出力操作を行う部分．OS はこの BIOS 経由でハードウェアに接するので，ハードウェアの差はこの BIOS で吸収される．そのため，同じソフトが動くパソコンでも BIOS の内容は異なる．

疑似命令：アセンブラ・システムでソース・プログラム中に記述されるが具体的な命令コードに変換されないで，アセンブル時の条件付け，変数の定義，タイトルの表示，リスト表示などのコントロールを行う．アセンブラ・システムに対する命令．

エディタ：ソース・プログラムを作成するためのユーティリティとして使われるようになった．その後ソース・プログラムの作成以外に文書作成を援助するための各種の機能が追加され，ワープロのようにディスプレイ画面で処理ができるようになり，文書作成に多く利用されるようになった．米国で使われるパソコンには，このエディタがワープロとして利用されることが多い．

高級言語：高水準言語とも呼び，特定の計算機システムに依存せず日常の言語に近いかたちの命令で，プログラムが書ける言語．高級言語の 1 命令は実際の計算機の命令を複数実行して実現する．高級言語で書かれたソース・プログラムを各言語システムが翻訳(コンパイル)する．Fortran, Pascal, C など多数の高級言語システムがパーソナル・コンピュータ用に提供されている．

新連載　第1回

C言語による実践プログラミング技術入門

ANSIのCと標準関数の詳細

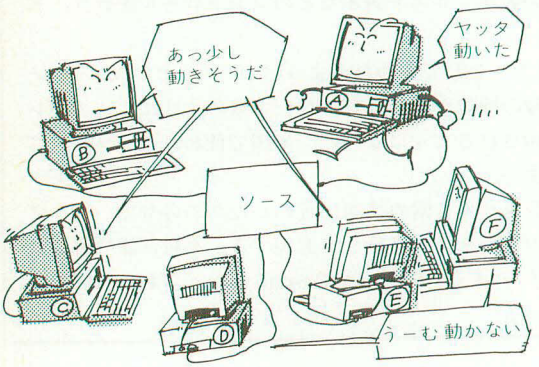
大貫 広幸・おおぬき ひろゆき

C言語は，アセンブラに近い高級言語として，開発技術者には必須のプログラミング言語になっています．また，最近ではソフトウェアについての解説書も数多く出版されていますが，その中でもC言語に関するものは特に多いように思います．

しかし，そのほとんどが入門書であり，各メーカのCコンパイラに関するコマンドの解説が主となっているようで，もう少し具体的な使い方やシステムを設計するときに必要なことまではほとんど説明されていません．

そこで，C言語の文法をマスタされ，ランタイム・ライブラリで定義されている関数やマクロもある程度は使いこなせるといった中級者の方を対象にして，少し具体的なプログラミング技術を解説してみようと思います．すなわち，ANSIが定義しているランタイム・ライブラリ上の関数やマクロの詳しい使い方，その関数やマクロがどのように作られていて，その元になるアルゴリズムにはどのようなものがあるのかといったことも紹介しながら，C言語のランタイム・ライブラリについてさらに理解を深めてもらい，プログラミングに役立ててもらおうというものです．

C言語の場合，ほかの多くの言語，たとえばBasicやPascal，FortranやCobol…といった言語とは異なり，文法では入出力や演算関数に関する規定はなく，C言語の文法ではデータの型や式，演算子，そしてフロー制御文といった構文やプリプロセッサによる前処理についてのみを規定し，入出力関係や算術関数，プロセス管理，記憶領域の割り当てなどの機能は，すべてライブラリ関数やマクロで処理するようになっています．



そのため，C言語を使ってプログラミングを行おうとした場合，C言語の文法を理解しているほかに，これらライブラリ関数やマクロの使い方を理解することが不可欠なわけです．

● パソコンやUNIXのC言語の互換性が問題

しかし，パーソナル・コンピュータ上で使われていた従来のC言語では，文法についてはC言語そしてUNIXオペレーティング・システムの産みの親であるB. W. Kernighan，D. M. Ritchie両氏により書かれたC言語の解説書である“The C programming language”(米国Prentice-Hall社発行)に書かれていた“APPENDIX A : C REFERENCE MANUAL”で定義されていた構文が一般に使われます．ライブラリについてはUNIX上で定義されていた関数やマクロを元にして，Cコンパイラの開発メーカが独自に関数やマクロを追加しています．

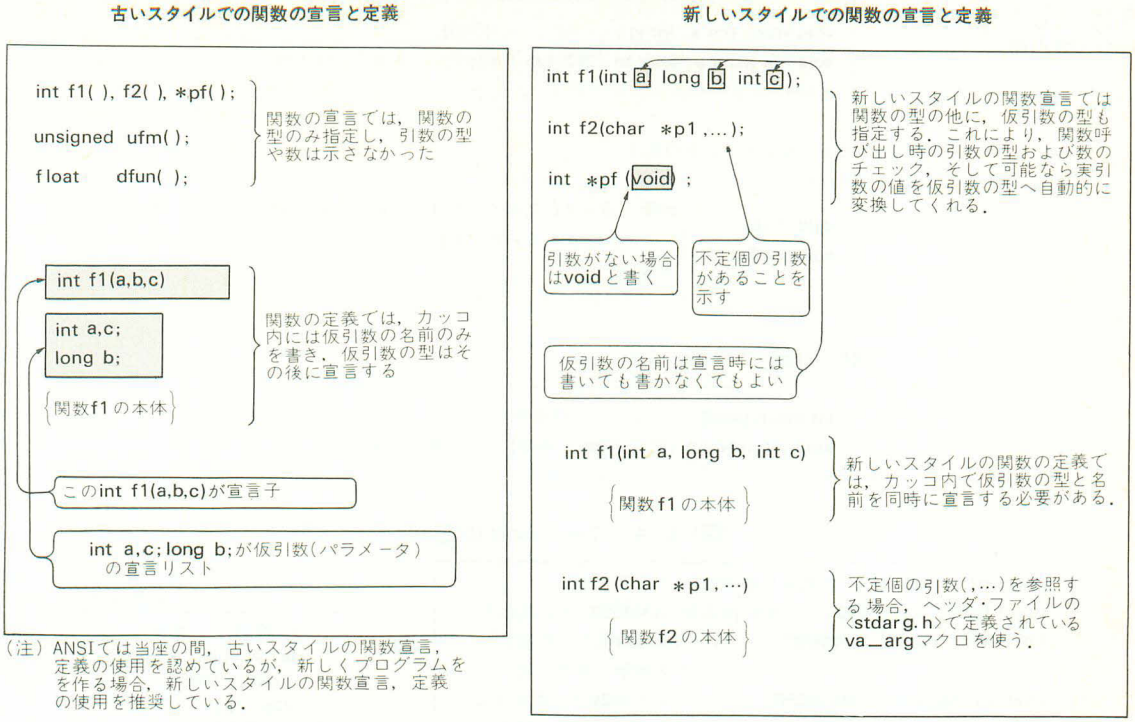
したがって，同じ機能をする関数やマクロでも，コンパイラ・メーカ間で名前や仕様が異なっていたり，UNIX準拠のライブラリであっても細かい点で動作が違っているなど，同じC言語でありながらメーカ間での互換性がとれていません．たとえば，A社のCコンパイラで作成したプログラムがB社のCコンパイラではコンパイルできなかったり，コンパイルできても実行形式のプログラムが正しく実行されないなど，使用するコンパイラを変えると，そのコンパイラに合わせてプログラムの一部修正が必要な場合が多く見受けられました．

● ANSIのC言語で統一化を進める

このようなメーカ間の互換性の問題がありながらも，パーソナル・コンピュータの普及とともにC言語が多く使われるようになり，規格として統一されたC言語の必要性が高まってきました．ANSI(米国国家規格協会)の標準化委員会によりC言語の規格化作業が行われ，昨年の秋(1988年10月)にプログラム言語としての最終的な規格案が規格(プログラム言語C，X3.159-1988)としての承認を求めANSIに提出されました．

そして，パーソナル・コンピュータ用の多くのCコ

〈図1-1〉ANSIでの関数の宣言と定義



（注）ANSIでは当座の間，古いスタイルの関数宣言，定義の使用を認めているが，新しくプログラムを作る場合，新しいスタイルの関数宣言，定義の使用を推奨している．

ンパイラもこのANSIの規格案を元に改良が加えられ，現在発売されている多くのCコンパイラは，従来からの仕様も残しつつ，ANSIの規格案に準拠するようになってきています．

これにより，我々ユーザはANSIの規格案に沿ってプログラミングを行えば，メーカに左右されることなく，違うメーカのCコンパイラであっても共通にコンパイル，実行可能なプログラムを作成できるようになったわけです(*1)．

このようなことから，この連載ではコンピュータの機種やオペレーティング・システムに依存する部分が少ないANSIの規格案のライブラリについて話を進めることにしました．また，掲載するプログラム例なども極力特定のコンパイラを意識しないように作るつもりでいますが，到底すべてのコンパイラで共通にコンパイルおよび実行できるプログラムなどは作れないため，最低下記の条件で動作するプログラム例を掲載するつもりでいます．

▶機種

NEC PC9801シリーズ

IBM PC/XT/ATおよびXT/ATのコンパチブル機

▶オペレーティング・システム

MS-DOS Ver 3.3 ……(PC9801シリーズ用)

MS-DOS Ver 4.01……(IBM PC/XT/AT用)

▶Cコンパイラ

マイクロソフト社のMS-C Ver 5.X
およびQuick C Ver 1.X

ボーランド社のTurbo C Ver 2.0

▶int，unsigned型のサイズとポインタのサイズ

ここでは，MS-CやTurbo Cが標準でサポートしているメモリ・モデルであるスモール・モデルでプログラム例を作成します．そのため，int, unsigned型のサイズは16ビット長とし，ポインタのサイズも16ビット長とします．

ANSIのC言語標準規格案について

このように現在，C言語には従来から使われていたCとANSIの規格案が規定しているCの二種類が存在するわけです．そのため，ANSIの規格案のCと従

(*1) これはパーソナル・コンピュータのみを使うユーザには関係ないことですが，現在UNIX上のCコンパイラについてはANSIのC言語仕様がまだ正式な規格でないということからか，従来からの言語仕様のものがほとんどで，ANSIのC言語の新しい仕様で書いたプログラムは，UNIX上のCコンパイラではコンパイルできないということがあります．近い将来UNIX上のCコンパイラの多くがANSI仕様を取り入れるものと思われますが，それまでこのような混乱が続くのではないのでしょうか．ですからUNIX，パーソナル・コンピュータ共通に使えるプログラムを作成する場合，逆にANSIの新しい仕様でプログラミングしないほうが現状はよいようです．

〈図1-2〉
キーワード
void の使用例

① 値を返さない関数の宣言，定義

```
void vfunc (int x, int y) ; ……宣言での使用例
void fun (long a, char b) {関数 fun の本体} ……定義での使用例
```

② 型をもたないポインタの宣言

```
void *vptr ;
void *f (int x) ;
```

void で宣言されたポインタはアドレスの記憶場所としてしか使用できない．そのため，void ポインタによるデータのアクセスやアドレス計算は不可(エラー)となる

③ 関数の引数がない場合の宣言，定義

```
int xvfun (void) ; ……宣言での使用例
long yvfun (void) {関数 yvfun の本体} ……定義での使用例
```

〈図1-3〉キーワード const の使用例

```
const int k=1000 ; ……変数 k の値 (1000) を定数化する
const float pi=3.141592 ; ……変数 pi の値 (3.141592) を定数化する
char const *st1p="abcdefg" ; ……文字列 abcdefg を定数化する．
                                ポインタ st1p は変更可能
char *const st2p="ABCDEFG" ; ……ポインタ st2p を定数化する．
                                文字列 ABCDEFG は変更可能
```

左の宣言に対して次のような文をコンパイルするとエラーまたは警告が出る

```
i=10 * ++k/u ;
pi=3.14 ;
*(st1p+1)='2' ;
st2p="xyz" ;
```

来から使われていたCを区別する意味で，従来から使われていたC，つまり“The C programming language”が規定していたC(*2)のことを筆者の頭文字を取り「K&RのC」と略して呼び，ANSIの規格案と区別することが多いようです．

ここでは，ANSIのCが従来からのK&RのCとどのような点で違っているのか，文法とライブラリに分けて見てみることにします．

● K&RのCとANSIのCとの文法上の相異点

ANSIのCは，文法的には従来のK&RのCの文法に対して，下記のような変更，追加，削除が行われ，K&RのCが持っていた文法の定義としてのあいまいさや時代遅れの面，そして使用上プログラム・ミスを起こしやすかった構文などが改められました．

これにより，K&RのCでは多々発生していた型の不一致によるデータを読み間違えやデータの破壊，関数へ渡す引数の個数を間違えるといった文法上のあいまいさからくるプログラムの動作不良といったものがANSIのCではコンパイル段階でチェックされ，弾かれるようになりました．しかし，このK&RのCの文法上のあいまいさを逆手にとり，効率のよい短いプログラムを書く，Cのベテラン・プログラマにとっては，ANSIのCは制約が多く嫌だという意見もあります．

筆者個人としては，従来のK&RのCではOKだった文がANSIのCでは警告やエラーになるのは面白くありません．しかし，ANSIのCではコンパイル段階でチェックが厳しくなり，プログラミングやコーディング段階のミスが途中で発見できるようになったことはデバッグのことも考え，ANSIのCのほうがメリットが多いように思います．

K&RのCに対してANSIのCで変更，追加，削除が行われた内容をまとめると，次のようになります．

① プロトタイプ宣言の機能が導入された．

プロトタイプ宣言の機能により，従来あいまいだった関数の実引数と仮引数の型や個数のチェックがコンパイラで行えるようになった．ただし，これにより従来あいまいさゆえに使えた実引数の数が可変な関数の定義ができなくなったため，新たに仮引数の宣言で不定個の引数を表す省略記法“,...”と，引数がない場合の指定 void が作られた(図1-1参照)．

② 識別子の長さは少なくとも最初の31文字までは有効にする．

これにより，関数名や変数名などの識別子は，少なくとも最初の31文字までは有効な名前と識別するよ

(*2) K&Rにより書かれた本“The C programming language”も昨年(1988年)に発売された第2版では，ANSIの規格案に準拠する形で大きく書き直され，ANSIのCを解説した本としては初めてのものとなりました．ANSIのCについての詳細を知りたい方はこの本を読まれることをお薦めします．ただし，ここでいっているK&RのCとは，1978年に発売された“The C programming language”の第1版のことを指しているものとします．

〈図1-4〉キーワード volatile の使用例

```
volatile unsigned intf ;
void intwait (void)
{
        intf=0 ;
        while(intf==0) ;
}
```

この例は，割り込みにより変数 **intf** の値が **0** 以外の値に設定されるのを待つ関数を記述したもの

〈図1-6〉l，u，ul，f を使用した例

```
0u 23u 376U      ……unsigned 型の定数
45l 253645437L   ……long 型の定数
876ul 366366UL   ……unsigned long 型の定数
1.45F 2.56e34f   ……float 型の定数
1.45l 6.5645e-12L……double 型の定数
```

〈図1-5〉列挙型の宣言の例

```
enum etg {K0, K1, K2, K3, K6=6, K8=8, K9}x=K8, y ; ……enum による列挙型変数 x, y の宣言例．etg は列挙タグ
enum etg z ;……列挙タグにより列挙型変数 z の宣言
```

上記の列挙型変数の宣言は **enum** を使わず# **define** と **int** 型変数の宣言で行うと次のように書くことができる
(ただし，これは≒ということである)

```
#define   K0   0
#define   K1   1
#define   K2   2
#define   K3   3
#define   K6   6
#define   K8   8
#define   K9   9
int x=K8, y ;
int z ;
```

うになった．

③ 新しいキーワードとして，**signed**，**void**，**const**，**volatile**，**enum** が追加された．

▶ **signed** は，符号付き整数を定義，あるいは宣言するときに使用するキーワード．

▶ **void** は，型のない値(つまり，値として意味を持たないデータ)を定義，あるいは宣言するもので，値を返さない関数を定義(宣言)するときや，示しているアドレスの型が不明なアドレスの記憶場所としてのみ使用するポインタを宣言するときに，**void** のキーワードを使う．また，関数の引数がないことを明確に宣言するのにも **void** を使う(図1-2 参照)．

▶ **const** は，変数や配列を宣言するとき，その値(宣言時に指定した初期値)を定数化するときに使用するキーワードで，**const** が指定された変数や配列をプログラムで変更しようとした場合，コンパイラはエラーまたは警告を出すようになる(図1-3 参照)．

▶ **volatile** は，宣言した変数や配列がほかのプロセス(ハードウェア割り込みを含む並行に走っているプログラム)から変更される可能性がある共通変数(配列)であることを宣言するためのキーワード．**volatile** で宣言された変数や配列は，コンパイラの最適化(CPU のレジスタ上に中間結果があれば，メモリ上の値ではなく，このレジスタ上の値を使用するようなオブジェクト・コードを生成する)が行われなくなり，アクセスのとき必ずメモリ上の値を使用するようなオブジェクト・コードが作られる(図1-4 参照)．

▶ **enum** は，列挙型変数を宣言するもので，**enum** を使うことで **int** 型変数の宣言とこの **int** 型変数に格納する複数の整数定数の名前の定義を一回で行うことができる．書き方は構造体や共用体の定義と似ている(図1-5 参照)．

④ 新しい型として，**signed char**，**unsigned char**，**unsigned long**，**long double** が追加された．

⑤ 定数値の最後に型を表す文字として **l**，**u**，**ul**，**f** が使用できるようになった．

l は **long** 型または **double** 型，**u** は **unsigned** 型，**ul** は **unsigned long** 型，**f** は **float** 型の定数を示す．**L**，**U**，**UL**，**F** の大文字も使える(図1-6 参照)．

⑥ エスケープ・シーケンスとしてベル・コード(\ **a**)，垂直タブ(\ **v**)，16 進数(\ **xhh**)が書けるようになった．

⑦ **entry** は予約語でなくなった．

entry は，従来予約語となっていたが，実際に使用さ

れることはなかったので予約語から外された.

⑧ 構造体や共用体のメンバの名前は，一意的な名前を持つ必要がなくなった.

これは，異なる構造体(共用体)であれば，ほかの構造体で使われている名前と同じメンバ名が使用できるということで，それにともなって，構造体や共用体に対して，ほかの構造体や共用体のメンバを指定することが許されなくなった.

⑨ 構造体や共用体に対する代入や関数への引数，関数からの戻り値として構造体や共用体の使用を許している.

⑩ プリプロセッサ命令として#**elif**(else-if の意味)が追加された.

また，#**if** や#**elif** の定数式の中でマクロが#**define** で定義されているか否かのチェックができるようになった.

マクロが定義されているか否かのチェックは，プリプロセッサの演算子 **defined** で行う．**defined** は，

defined 識別子

または，

defined (識別子)

と書き，識別子が事前に定義されていれば **1L**，定義されていなければ **0L** の定数となる.

⑪ マクロを定義する#**define** 中で，プリプロセッサの演算子#および##の演算が使えるようになった.

演算子#は，#の後に書かれたマクロの引数を文字列定数に変換する演算子で，##は##の左右の文字列を連結し，一つの文字列にする演算子(図1-7 参照).

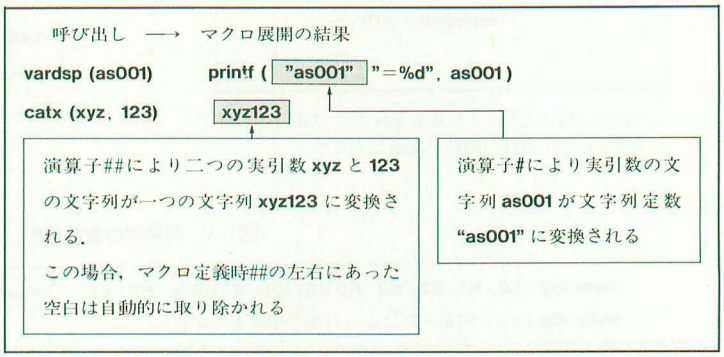
⑫ プリプロセッサ命令として，コンパイラにプログラム上から指示を与えるための命令#**pragma** と，プログラム上からコンパイル中にユーザ指定文字列を表示(エラー表示として)するための命令#**error** が追加された．ただし，#**pragma** の詳しい内容はコンパイラによって異なるので，注意して使わなければならない.

⑬ コンパイラは，定義済みの名前として図1-8 に示す五つの名前を自動的に定義する.

〈図1-7〉 プリプロセッサの演算子#および##の使い方

```
# define   vardsp (v)   printf (# v"=%d", v)
# define   catx (a,b)   (a   ##   b)
```

上記のマクロの呼び出しとその展開は次の例のようになる



〈図1-8〉 定義ずみの名前

__STDC__	コンパイラがANSI準拠でコンパイル中なら1，ANSI以外なら__STDC__の定義そのものがない
__FILE__	現在コンパイル中のファイル名
__LINE__	現在コンパイル中の行番号
__DATE__	コンパイル時の日付け(文字列)
__TIME__	コンパイル時間(文字列)

● ANSI 規格案のライブラリ

次に，ANSI 規格案のCで規定しているライブラリについてみてみます．一部追加修正はあるものの，ANSI 規格案のライブラリについてもUNIXのC言語ライブラリを元に，特定の機種やオペレーティング・システムに依存しない関数やマクロを選び，それを規格案としてのライブラリとして規定しています.

UNIX 上のライブラリは，大きく“システム・コール”と“サブルーチン&ライブラリ”の二種類に分けられ，“システム・コール”は，UNIX の機能を直接呼び出すもので，MS-DOS のファンクション・コールに相当するものです．ただし，MS-DOS のファンクション・コールの場合は，アセンブラから呼び出されることを前提としてパラメータなどが決められていましたが，UNIX の場合システム・コールもC言語から呼び出されることを前提にしているため，Cの関数として定義されているわけです.

システム・コールで定義されている関数は，基本的にはUNIX のカーネルがユーザにサービスしている機能を直接呼び出しているに過ぎないため，バイト単位の入出力やプロセス管理，記憶領域の管理といったどちらかというと低レベルの機能を提供しています.

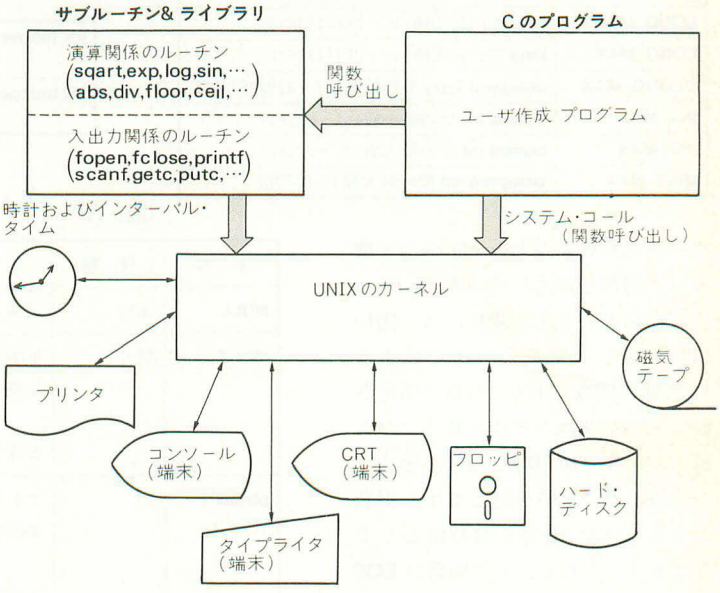
表1-1 はシステム・コールのうち代表的な関数を抜粋し一覧表にしたものです．これら“システム・コール”上の関数やマクロを正しく使用するためには，自分が使用しているUNIX のカーネルに関する知識が多少必要となります.

〈表1-1〉UNIXのシステム・コール(抜粋)

関数名	機　　能
brk	データ・セグメントの上限を設定
sbrk	データ・セグメントの大きさを変更
chdir	ワーク・ディレクトリの変更
open	ファイルのオープン
creat	ファイルの作成とオープン
read	ファイルからのリード
write	ファイルへのライト
lseek	ファイルより次にリード/ライトする位置を変更
close	ファイルのクローズ
unlink	ファイルの消去
mknod	ディレクトリ，スペシャル・ファイル，通常ファイルの作成
fork	新しいプロセスを作成する
execl, execv, …	ファイルをロードし同一プロセスで実行
exit	プロセスの終了

関数名	機　　能
wait	**fork** で作成したプロセスが終了するのを待つ
kill	プロセスへシグナルを送信する
signal	シグナル受信の動作を指定する
pause	シグナル発生までのプロセスを停止する
nice	プロセスの優先度の設定
pipe	プロセス間通信用バッファを作成する
profil	プロセス実行時間の計測
ptrace	プロセスのトレース，デバッグで使用
alarm	プロセスのアラーム・クロックを設定する
access	ファイルのアクセス・モードを設定
fstat	ファイル・ステータスの読み出し
chmod	ファイル・モードの変更
getpid	プロセスIDを読み出す
getppid	親プロセスのIDを読む
sync	スーパ・ブロックの更新
time	現在の時刻をグリニッジ標準時1970年1月1日00：00：00からの秒数で求める

〈図1-9〉UNIXの関数“システム・コール”と“サブルーチン＆ライブラリ”の関係



それに比べ，“サブルーチン＆ライブラリ”というのは，元々C言語でプログラムを作るうえで，あると便利と思われる関数やマクロを事前に作り，ユーザに提供したもので，日頃よく使われる数値や文字列単位の入出力，算術関数や文字，文字列操作といったルーチンを提供しています．

サブルーチン＆ライブラリで定義されている関数は**図1-9**のように，直接UNIXのカーネルを呼び出さないため，オペレーティング・システムのカーネルの機能に直接左右されることがあまりありません．そこで，関数やマクロの呼び出し手順を合わせることによりUNIX以外の実行環境でも容易に実現できる関数やマクロが多いことから，ANSI規格案の関数やマクロもこの“サブルーチン＆ライブラリ”上から多く選び出されています．

ANSIの規格案で定義している関数やマクロは，機能別に分類され，**図1-10**に示す15種類のヘッダ・ファイルに分けられ宣言されています．

ですから，ユーザ・プログラムでANSI規格案の関数やマクロを使用する場合，ソース・プログラムの先頭で，プリプロセッサの#**include**命令を使い，

#**include**〈ヘッダ・ファイル名〉

とヘッダ・ファイルを読み込んでおくことで，関数やマクロを使用するうえで必要なプロトタイプ宣言や外部変数宣言，#**define**でのマクロの定義などを行うことができます．

個々のヘッダ・ファイルの内容については，次のようになっています．

① **stddef.h**(よく使われる定数や変数，型やマクロの定義)

ヘッダ・ファイル**stddef.h**は，ライブラリ上の関数やマクロを使用するうえで，よく使用される定数や変数，型の名前を定義しているファイルです．このファイルで，定義されている名前は**図1-11**に示した4種類です．

② **limits.h**(整数型の範囲)

ヘッダ・ファイル**limits.h**は，整数型の値の範囲を示す定数を定義しているファイルで，**図1-12**に示す15種類の名前が定義されています．カッコ内の値はヘッダ・ファイルに定義されている範囲の一例を示したものです．

③ **ctype.h**(文字クラスのテストと変換)

ヘッダ・ファイル**ctype.h**は，文字をテストするた

〈図1-10〉ANSIに定義されている関数とマクロ

stddef.h	limits.h	ctype.h	string.h	stdarg.h
stdio.h	stdlib.h	time.h	float.h	math.h
signal.h	setjmp.h	assert.h	locale.h	error.h

〈図1-12〉ヘッダ・ファイルlimits.hに定義されている定数の名前

定数の名前	内容と範囲
CHAR_BIT	char型のビット数(=8)
CHAR_MIN	char型の最小値(=−127)
CHAR_MAX	char型の最大値(=+127)
SCHAR_MIN	signed char型の最小値(=−127)
SCHAR_MAX	signed char型の最大値(=+127)
UCHAR_MAX	unsigned char型の最大値(=255)
SHRT_MIN	short型の最小値(=−32767)
SHRT_MAX	short型の最大値(=+32767)
USHRT_MAX	unsigned short型の最大値(=65535)
LONG_MIN	long型の最小値(=−2147483647)
LONG_MAX	long型の最大値(=+2147483647)
ULONG_MAX	unsigned long型の最大値(=4294967295)
INT_MIN	signed int型の最小値(=−32767)
INT_MAX	signed int型の最大値(=+32767)
UINT_MAX	unsigned int型の最大値(=65535)

〈図1-13〉ヘッダ・ファイルctype.hに宣言されている関数

関数	内容
isdigit (c)	文字cは10進数を構成する文字なのかテストする
isxdigit (c)	文字cは16進数を構成する文字なのかテストする
islower (c)	文字cは英字の小文字なのかテストする
isupper (c)	文字cは英字の大文字なのかテストする
isalpha (c)	文字cは英字なのかテストする
isalnum (c)	文字cは英数字なのかテストする
iscntrl (c)	文字cは制御文字なのかテストする
isgraph (c)	文字cは空白以外の印字可能文字なのかテストする
isprint (c)	文字cは空白も含めた印字可能文字なのかテストする
ispunct (c)	文字cは区切り文字なのかテストする
isspace (c)	文字cは空白，改行，復帰，改頁，水平タブ，垂直タブの何れなのかテストする

(a) 文字をテストする関数

int tolower (int c)	文字cが英字の大文字なら小文字に変換する
int toupper (int c)	文字cが英字の小文字なら大文字に変換する

(b) 文字の変換を行う関数

〈図1-11〉ヘッダ・ファイルstddef.hに登録されている名前

名称	種類	内容
NULL	定数	ヌル・ポインタ，#defineで定義された定数0
size_t	型	単項演算子sizeofが返す値の型．ヘッダ上ではtypedefを使い， typedef unsigned int size_t ; と宣言されている
ptrdiff_t	型	アドレス計算で，二つのポインタの差を取ったときの結果の型．ヘッダ上ではtypedefを使い， typedef int ptrdiff_t ; と宣言されている
errno	グローバル変数	ライブラリ上の関数を実行したとき，エラーが発生した場合，その詳細なエラー情報を番号で設定してくるグローバル変数．errnoは通常int型で，ヘッダ上では次のように宣言されている． extern int errno ;

めの関数や英字の大小文字への変換を行う関数を宣言しています．

文字をテストする関数には，図1-13(a)に示した関数名がisで始まる11種類の関数があり，引数で指定された文字が関数のテスト条件に合えばゼロでない値(真)を返し，そうでなければゼロ(偽)を返します．引数となる文字はint型の整数値として与えます．ただし，その引数はEOF(End Of Fileを表す値，stdio.hで定義されている定数の名前で，その値は−1)かunsigned char型の値の範囲内(0〜UCHAR_MAX)で表すことができる値でなければなりません．範囲外の値が引数として指定されていた場合，その結果は未定となります．

文字の変換を行う関数としては，図1-13(b)に示した関数名がtoで始まる二種類の関数があります．この二つの関数も引数の条件は文字テストの関数と同じで，EOFかあるいは0〜UCHAR_MAXの範囲の値である必要があります．これ以外の範囲を指定すると，その変換結果は未定となります．

④ string.h(文字列操作関数およびメモリ操作関数)

ヘッダ・ファイルstring.hは，文字列やメモリ上のデータを操作するための関数を集め，宣言しています．

文字列を操作するための関数には，図1-14(a)に示した関数名がstrで始まる15種類の関数があります．

メモリ上のデータを操作するための関数には，図1-14(b)に示した関数名がmemで始まる5種類の関数があります．

⑤ stdarg.h(可変引数リストをアクセスするためのマクロ)

ヘッダ・ファイルstdarg.hは，呼び出された関数側

〈図1-14〉ヘッダ・ファイル string.h に宣言されている関数

関　　数	内　　容
char *strcpy (char *s, const char *t)	文字列 t を s にコピーし，s を返す
char *strncpy (char *s, const char *t, size_t n)	文字列 t を s に n 文字分コピーし，s を返す．ただし，文字列 t の長さが n 以上の場合，文字列の終わりを示す'\0'は転送されないので注意
char *strcat (char *s, const char *t)	文字列 s の後ろに文字列 t を連結し，s を返す
char *strncat (char *s, const char *t, size_t n)	文字列 s の後ろに文字列 t を最大 n 文字連結し，s を返す
int strcmp (const char *s, const char *t)	文字列 s と文字列 t を比較し， 文字列 s＜文字列 t のときは負の値を返し， 文字列 s＝文字列 t のときはゼロ， 文字列 s＞文字列 t のときは正の値を返す
int strncmp (const char *s, const char *t, size_t n)	文字列 s と文字列 t を最大 n 文字分を比較する．比較結果は strcmp と同じ
char *strchr (const char *s, int c)	文字列 s の最初にある文字 c へのポインタを返す．文字 c がなければ NULL を返してくる
char *strrchr (const char *s, int c)	文字列 s の最後にある文字 c へのポインタを返す．文字 c がなければ NULL を返してくる
size_t sirspn (const char *s, const char *t)	文字列 t に含まれている文字のみで構成されている文字列 s の先頭の部分の長さを返す
size_t strcspn (const char *s, const char *t)	文字列 t に含まれない文字のみで構成されている文字列 s の先頭の部分の文字数を返す
char *strpbrk (const char *s, const char *t)	文字列 t に含まれている文字が文字列 s の中で最初に出てくる位置へのポインタを返す．もし，文字列 t に含まれている文字が文字列 s になければ NULL を返してくる
char *strstr (const char *s, const char *t)	文字列 s 中で文字列 t が最初に出てくる位置へのポインタを返す．文字列 t がなければ NULL を返してくる
size_t strlen (const char *s)	文字列 s の長さを返す
char *strerror (int n)	エラー番号 n に対応する文字列(エラー・メッセージ)へのポインタを返してくる．エラー番号と文字列との対応は処理系に依存する
char *strtok (char *s, const char *t)	文字列 t の任意に文字を区切りとするトークンへのポインタを返してくる．トークンが発見できなかった場合は NULL が返されてくる． s に対して文字列を与えれば，その文字列 s に対してトークンの検索が行われるが，s に対して NULL を指定した場合，前回検索に使用した文字列に対して，次のトークンの検索が行われる

(a) 文字列を操作するための関数

関　　数	内　　容
void *memcpy (void *s, const void *t, size_t n)	メモリ上のデータ t を s へ n 文字コピーする
void *memmove (void *s, const void *t, size_t n)	メモリ上のデータ t を s へ n 文字コピーする．ただし，データ領域が重なっていてもデータを壊すことなく転送できるのが memcpy と異なるところ
int memcmp (const void *s, const void *t, size_t n)	メモリ上のデータ s と t を n 文字分比較する．比較結果は strcmp と同じ
void *memchr (const void *s, int c, size_t n)	メモリ上のデータ s から n バイト以内にあり，最初に発見された文字 c へのポインタを返す．文字 c がなければ NULL を返してくる
void *memset (void *s, int c, size_t n)	s から n 文字分の領域を文字 c で埋める

(b) メモリ上のデータを操作する関数

で不定個の引数(, . . .)をアクセス(参照のみ)するために用意されたマクロを定義しています．定義されているマクロは図1-15 に示した三種類です．

図中の名前が va_ で始まるマクロを使用する場合，各引数を順に示すための型 va_list を使い変数 ap を，

```
va_list ap ;
```

と宣言しておく必要があります(ここでは，ap という変数名を使用したが，実際に使用する場合はどのような名前でもかまわない)．型 va_list は stdarg.h 上で宣言されている型です．

〈図1-15〉ヘッダ・ファイル stdarg.h に定義されているマクロ

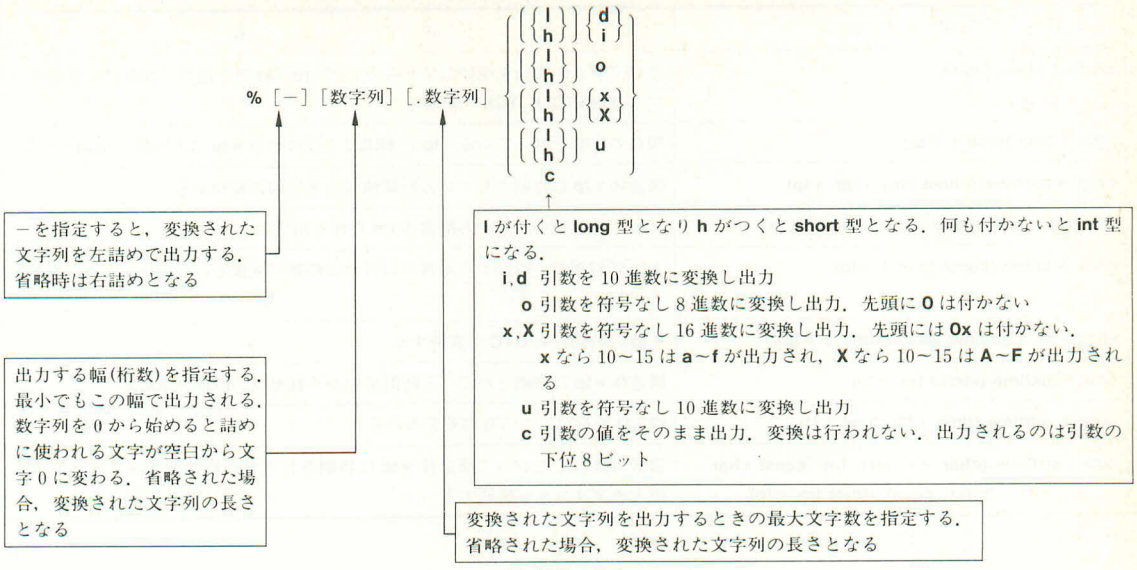
マ ク ロ 名	内　　容
va_start (va_list ap, last)	可変引数の取り出し処理開始
va_arg (va_list ap, type)	次の可変引数の取り出し
va_end (va_list ap)	可変引数の取り出し処理終了

(注)　ここで，ap は後述する変数の名前で，last は引数リストの最後に書かれた有効な引数の名前，たとえば fn(a,b,c,d,. . .)と引数が宣言されていれば last に書かれる引数名は d となる．type は取り出す引数の型名を示す．

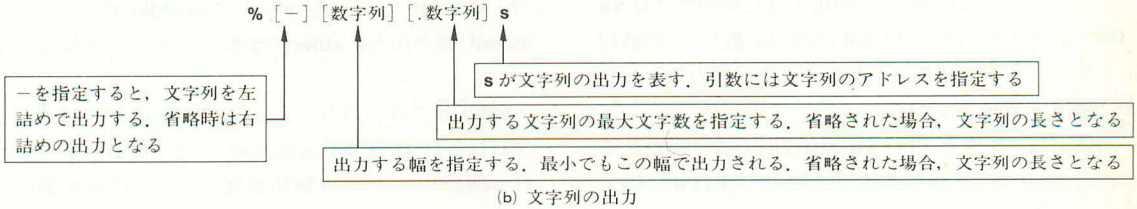
〈図1-16〉ヘッダ・ファイル stdio.h に宣言されている関数(抜粋)

関　　数	内　　容
FILE ＊fopen (const char ＊filename, const char ＊mode)	**filename** で指定したファイルやデバイスを，指定した **mode** でオープンする．オープンが成功すればストリームが返され，失敗すれば **NULL** が返されてくる
FILE ＊freopen (const char ＊filename, const char ＊mode, FILE ＊stream)	**filename** で指定したファイルやデバイスを，指定した **mode** でオープンし，指定した **stream** に割り当てる
FILE ＊tmpfile (void)	一時ファイルを作成し，そのストリームを返してくる．作成に失敗した場合 **NULL** が返されてくる．一時ファイルはクローズ時自動的に削除される
int fclose (FILE ＊stream)	指定した **stream** をクローズする．クローズに成功するとゼロ，失敗すると **EOF** が返される
int remove (const char ＊filename)	**filename** で指定したファイルを削除する．削除できた場合はゼロ，失敗した場合はゼロ以外の値が返されてくる
int rename (const char ＊oldfn, const char ＊newfn)	ファイル名を **oldfn** から **newfn** に変更する．変更に成功すればゼロ，失敗した場合はゼロ以外の値が返されてくる
int fprintf (FILE ＊stream, const char ＊fmt,...)	指定したストリームに対して，書式 **fmt** にしたがって可変引数リスト上の引数の値を文字列に変換し出力する．書式 **fmt** が示す文字列中，文字**%**で始まるトークンを書式指定に使用している．**%**で始まる書式指定以外の文字はそのまま出力される．**fprintf** は出力に成功すれば出力した文字数が返され，エラーした場合は負の値が返される
int printf (const char ＊fmt,...)	標準出力へ **fprintf** と同様な書式変換出力を行う
int sprintf (char ＊s, const char ＊fmt,...)	**fprintf** と同様に書式変換を行うが，出力先がメモリ上の文字列となっている
int fscanf (FILE ＊stream, const char ＊fmt,...)	指定したストリームから文字列を入力し，書式 **fmt** にしたがって可変引数リスト上の引数の値(内容はポインタ)の書式にしたがいデータ変換し格納する．書式 **fmt** が示す文字列中，文字**%**で始まるトークンを書式指定に使用している．空白を除く**%**で始まる書式指定以外の文字は，入力された文字に対して常に一致が取られる．もし，ここで一致しない文字を発見した場合 **fscanf** の実行は中止され，残りの文字列は次の入力にまわされる．**fscanf** は入力に成功すれば変換，格納した項目数が返され，エラーあるいはファイルが終わりに達したときは **EOF** が返されてくる
int scanf (const char ＊fmt,...)	標準入力から，**fscanf** と同様な書式変換入力を行う
int sscanf (char ＊s, const char ＊fmt,...)	指定した文字列から，**fscanf** と同様な書式変換入力を行う
int fputc (int c, FILE ＊stream)	指定したストリームへ文字 **c** を出力する．エラーなら **EOF**，成功なら非負の値を返す
int putc (int c, FILE ＊stream)	指定したストリームへ文字 **c** を出力する．**putc** はマクロで定義されていること以外，動作的には **fputc** と同じ
int putchar (char c)	標準出力へ文字 **c** を出力する．エラーなら **EOF**，成功なら非負の値を返す
int fputs (const char ＊s, FILE ＊stream)	指定したストリームへ文字列 **s** を出力する．エラーなら **EOF**，成功なら非負の値を返す
int puts (const char ＊s)	標準出力へ文字列 **s** を出力し，改行を行う．エラーなら **EOF**，成功なら非負の値を返す
int fgetc (FILE ＊stream)	指定したストリームから文字を入力する．エラーまたはファイルの終わりなら **EOF** を返す
int getc (FILE ＊stream)	指定したストリームから文字を入力する．**getc** はマクロで定義されていること以外，動作的には **fgetc** と同じ
int getchar (void)	標準入力から文字を入力する．エラーまたはファイルの終わりなら **EOF** を返す
char ＊fgets (char ＊s, int n, FILE ＊stream)	**fgets** の入力は，指定したストリームから(n-1)文字入力するか改行が入力されるまで行われる．入力に成功すれば **s** を返すが，エラーまたはファイルの終わりなら **NULL** を返す
char ＊gets (char ＊s)	標準入力から一行分の文字列入力を行う．入力に成功すれば **s** を返すが，エラーまたはファイルの終わりなら **NULL** を返す
size_t fread (void ＊p, size_t size, size_t n, FILE ＊stream)	指定したストリームから，**size** の大きさを持つデータを **n** 個分，配列 **p** に入力してくる．**fread** が返す値は実際に入力したデータの個数，もし，返された個数が **n** より小さい場合，エラーまたはファイルの終わりに達したかの何れかである
size_t fwrite (const void ＊p, size_t size, size_t n, FILE ＊stream)	指定したストリームへ，1データ **size** の大きさを持つ配列 **p** のデータを **n** 個分出力する．**fwite** が返す値は実際に出力したデータの個数．もし，返された個数が **n** より小さい場合，出力でエラーが発生したことになる
int fseek (FILE ＊stream, long offset, int org)	指定したストリームの R/W ポインタの位置を **org** をベースとして **offset** 文字分移動する．**org** には，**SEEK_SET**(ファイルの先頭)，**SEEK_CUR**(現在値)，**SEEK_END**(ファイルの終わり)を指定する．エラーの場合，ゼロでない値が返されてくる
void rewind (FILE ＊stream)	指定したストリームの R/W ポインタの位置をファイルの先頭に持って行く
long ftall (FILE ＊stream)	指定したストリームの R/W ポインタの位置を読み出してくる．エラーの場合−**1L** を返してくる
int feof (FILE ＊stream)	指定したストリームの R/W ポインタの位置が，ファイルの終わりに達したか調べる．終わりに達していればゼロ以外の値が返されてくる
int ferror (FILE ＊stream)	指定したストリームにエラーが発生しているか調べる．エラーが発生していればゼロ以外の値が返されてくる
void clearerr (FILE ＊stream)	指定したストリームのエラーおよび終わりに達したことを表すフラグをクリアするための関数

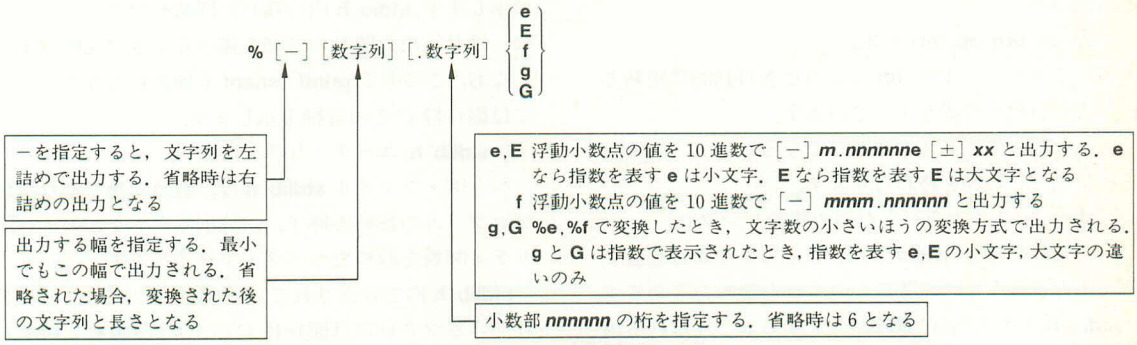
〈図1-17(a)〉 fprintf，printf，sprintf で使われる主な書式



(a) 整数の出力

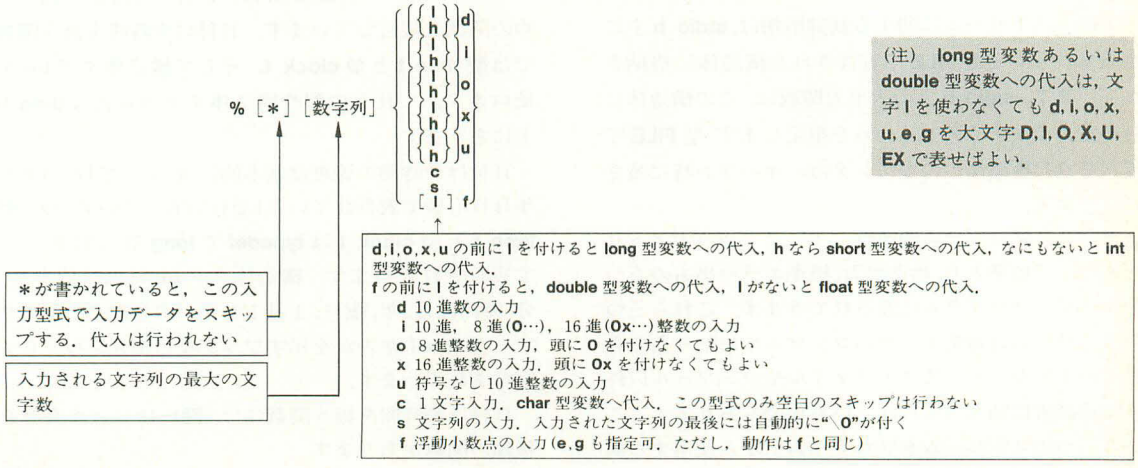


(b) 文字列の出力



(c) 浮動小数点の出力

〈図1-17(b)〉 fscanf，scanf，sscanf で使われる主な書式



(注) long 型変数あるいは double 型変数への代入は，文字 l を使わなくても d, i, o, x, u, e, g を大文字 D, I, O, X, U, EX で表せばよい．

〈図1-19〉ヘッダ・ファイル time.h に宣言されている関数

関　　数	内　　容
clock_t clock (void)	プログラムが実行を開始してからの CPU 使用時間を返す．秒単位に変換するには clock/CLK_TCK の計算を行う
time_t time (time_t ＊tp)	現在の時間を返してくる．tp が NULL でなければ＊tp にも時間が格納される
char ＊asctime (const struct tm ＊tp)	構造体＊tp に格納されている時間情報を文字列に変換する
struct tm ＊localtime (const time_t ＊tp)	＊tp の秒単位の時間から構造体 tm を作り出す
char ＊ctime (const time_t ＊tp)	＊tp の秒単位の時間から通常の日付けと時間に変換し，それを文字列に変換する
struct tm ＊gmtime (const time_t ＊tp)	＊tp の時間から UTC へ変換する
time_t mktime (strcut tm ＊tp)	構造体＊tp に格納されている時間情報から秒単位の時間を求める
double difftime (time_t t2, time_t t1)	t2−t1 を行い二つの時間差を求める
size_t strftime (char ＊s, size_t n, const char ＊fmt, const struct tm ＊tp)	書式 fmt にしたがって構造体＊tp に格納されている時間情報を文字へ変換し最大 n 文字分 s へ格納する

そのうえで，変数 ap を初期化するためのマクロ va_start を呼び出し，仮引数 last の次の引数(,...の最初の引数)の位置を ap にセットし，可変引数のアクセスの準備をします．

可変引数のアクセスは，va_arg で行います．たとえば引数が int 型で，その値に 3 を掛けて変数 u に代入する式の場合，

```
u=av_arg(ap,int) * 3 ;
```

と書くことができます．ap はこのとき自動的に更新され，次の引数の位置を示しています．

可変引数のアクセスが必要なくなった場合，va_end を呼び出し，処理を終わらせます．

⑥ stdio.h(入出力を行うための関数やマクロ)

ヘッダ・ファイル stdio.h は，ディスクや周辺装置との入出力を行うための関数やマクロを集めたものです．

stdio.h 上の入出力関数は，ストリームと呼ばれる論理的なデータ構造に対して行われます．実際のディスク上のファイルや周辺装置との対応はオープンにより行われ，切り離しはクローズによって行われます．

実際のストリームに関する制御情報は，stdio.h 上に定義されている型 FILE で宣言された構造体に格納されています．stdio.h 上の入出力関数は，この構造体に対するポインタでストリームを指定します．型 FILE で宣言された構造体へのポインタは，オープン時に渡されてきます．

ただし，プログラム開始時，オープン済みのストリームとして標準入力，標準出力，標準エラー出力の三つがユーザ・プログラムに渡されてきます．これら三つのストリームは通常ユーザのコンソールに割り当てられていますが，ディスク・ファイルやコンソール以外の周辺装置に割り当てられている場合もあります．これら三つのストリームをプログラム上から使用する場合，stdio.h 上で定義済みの名前 stdin(標準入力)，stdout(標準出力)，stderr(標準エラー出力)を使用します．

stdio.h に宣言されている関数や定義されているマクロは合計 41 種類もあるため，ここでは通常よく使われる関数やマクロ 30 種類を選び，その内容を図1-16 に示します．stdio.h 内の残りの関数やマクロについては，後日入出力関数の詳細を述べるときに説明します．

なお，この中で printf，scanf で使われる書式については図1-17 にその詳細を示します．

⑦ stdlib.h(ユーティリティ関数)

ヘッダ・ファイル stdlib.h は，数値変換や整数関数，プログラムの起動や終了，記憶割り当てなどのユーティリティ関数を収めたヘッダ・ファイルです．

stdlib.h 内で宣言されている関数，ならびに定義されているマクロには図1-18 に示す 23 種類のものがあります．

⑧ time.h(日付け，時刻関数)

ヘッダ・ファイル time.h は，日付けや時間を扱うための関数を宣言しています．日付けや時間を扱う関数では型 time_t と型 clock_t，そして構造体タグ tm を使います．これらの型や構造体タグの宣言は time.h 上にあります．

日付けや時間の管理は基本的には，すべて秒(つまり年月日も秒で表されている)で行い表しているため，型 time_t と型 clock_t は typedef で long 型の別名として宣言されています．構造体タグ tm には時間を秒，分，時，日，月，年，曜日，1 月 1 日からの日数，そしてサマータイム中か否かを示すフラグなどがメンバとして宣言されています．

日付けや時間を扱う関数には，図1-19 に示す合計 9 種類の関数があります．

〈図1-18〉ヘッダ・ファイル stdlib.h に宣言されている関数

関　　数	内　　容
int atoi (const char *s)	文字列で表された値を int 型の値に変換する
long atol (const char *s)	文字列で表された値を long 型の値に変換する
double atof (const char *s)	文字列で表された値を double 型の値に変換する
long strtol (const char *s, char **ep, int base)	文字列の最初のトークンを base の値を基数として long 型の値に変換する．このとき，トークンの前の空白は無視される．また，ep が NULL でなければ，*ep には変換されなかった残りの文字列へのポインタが格納される．base にゼロを指定すると，基数は文字列上の最初の数字で判断され，0 で始まれば 8 進，0x あるいは 0X で始まれば 16 進と判断する．base に値を設定する場合 2～36 の値が設定できる．結果がオーバフローすると符号により LONG_MAX あるいは LONG_MIN が返される
unsigned long strtoul (const char *s, char **ep, int base)	結果が unsigned long 型で，オーバフローした場合，ULONG_MAX となること以外は strtol と同じ
double strtod (const char *s, char **ep)	文字列の最初のトークンを double 型の値に変換する．このとき，トークンの前の空白は無視される．また，ep が NULL でなければ，*ep には変換されなかった残りの文字列へのポインタが格納される．結果がオーバフローした場合，符号と共に math.h で定義されている定数 HUGE_VAL が返ってくる
int rand (void)	関数 rand は，0～RAND_MAX の範囲の整数値の疑似乱数を発生させる関数．RAND_MAX は stdlib.h 内の #define で定義されている値で，MS-C や Turbo-C では 32767 に設定されている
void srand (unsigned int seed)	疑似乱数の新しい系の種として seed を指定する
int abs (int x)　ならびに long labs (long x)	x の絶対値を取る
div_t div (int x, int y)　ならびに ldiv_t ldiv (long x, long y)	x÷y の商と余りを求める関数．除算結果は，型 div_t あるいは型 ldiv_t の構造体のメンバ quot および rem に格納される．型 div_t および型 ldiv_t は stdlib.h 内の typedef により次のように宣言されている． typedef struct {int quot; int rem} div_t; typedef struct {long quot; long rem} ldiv_t;
void abort (void)	プログラムを異常終了させる
void exit (int stat)	プログラムを正常終了させる
int atexit (void (*xf) (void))	exit でプログラムを正常終了させるとき，呼ばれるユーザ関数を登録する
int system (const char *cmd)	文字列 cmd で指定されたプログラムを起動する
char *getenv (const char *ename)	環境変数に文字列 ename と同じ名前のものがあれば，その環境変数の文字列へのポインタを返す．なければ，NULL が返される
void *calloc (size_t n, size_t size)	未使用の領域から n×size バイトのゼロ・クリアしてある領域を割り当ててもらい，その領域のポインタが返される．割り当てる領域がない場合 NULL が返される
void *malloc (size_t size)	未使用の領域から size バイトの領域を割り当ててもらい，その領域のポインタが返される．割り当てる領域がない場合 NULL が返される
void *realloc (void *p, size_t size)	割り当て済みの領域 p の大きさを size バイトに変える．再割り当てに成功すればその新領域へのポインタが返される．再割り当ての領域がない場合 NULL が返される
void free (void *p)	割り当ててもらった領域を解放する
void *bsearch (const void *key, const void *x, size_t n, size_t size, int (*cmp) (const void *, const void *))	bsearch は検索を行う関数で，配列 x［0］～X［n－1］中から *key と一致する項目を検索しそのポインタを返す．検索時に使用する比較は cmp で指定する
void qsort (void *x, size_t n, size_t size, int (*cmp) (const void *, const void *))	qsort は配列 x［0］～x［n－1］を昇順にソートする関数で，ソート時使用する比較は cmp で指定する

〈図1-20〉ヘッダ・ファイルfloat.hに定義されている定数の名前

型	定数の名前とその内容
float 型用	**FLT_RADIX** …基数 **FLT_DIG** …10進数にした場合の有効な桁数 **FLT_MANT_DIG**…仮数部の桁数 **FLT_MAX** …絶対値の最大 **FLT_MIN** …絶対値の最小 **FLT_MAX_EXP** …指数部の最大値 **FLT_MIN_EXP** …指数部の最小値 **FLT_EPSILON** …1.0+x≠1.0となる最小のxの値
double 型用	上記の **FLT_** が **DBL_** となっている
long double 型用	上記の **FLT_** が **LDBL_** となっている

⑨ **float.h**(浮動小数点演算に関する定数)

ヘッダ・ファイル **float.h** は，浮動小数点演算に使用される定数が定義されています．

定義されている主な定数の名前には**図1-20**に示すようなものがあります．定数の値については，システムが採用している浮動小数点演算の形式により異なるためここでは示しません．

⑩ **math.h**(数学関数)

ヘッダ・ファイル **math.h** には，各種の数学関数が宣言されています．**図1-21**にその内容を示します．三角関数の角度はラジアンを使用しています．

⑪ **signal.h**(シグナル)

ヘッダ・ファイル **signal.h** は，例外処理を行うため

〈図1-21〉ヘッダ・ファイルmath.hに宣言されている数学関数

関　　数	内　　容
double sin (double x)	sin **x**
double cos (double x)	cos **x**
double tan (double x)	tan **x**
double asin (double x)	$\sin^{-1}$ **x**
double acos (double x)	$\cos^{-1}$ **x**
double atan (double x)	$\tan^{-1}$ **x**
double atan2 (double x, double y)	$\tan^{-1}$ (**y/x**)
double sinh (double x)	sinh **x**
double cosh (double x)	cosh **x**
double tanh (double x)	tanh **x**
double exp (double x)	e^{x}
double log (double x)	$\log_e$ **x**
double log10 (double x)	$\log_{10}$ **x**
double pow (double x, double y)	x^{y}
double sqrt (double x)	$\sqrt{x}$
double ceil (double x)	**x**より小さくない最小の整数
double floor (double x)	**x**より大きくない最大の整数
double fabs (double x)	**x**の絶対値
double ldexp (double x, int n)	$x \times 2^{n}$
double frexp (double x, int *n)	仮数部と指数部に分ける．指数部は***n**に格納され仮数部のみが返されてくる
double modf (double x, double *ip)	整数部と小数部に分ける．整数部は***ip**に格納され，小数部が返されてくる
double fmod (double x, double y)	**x**÷**y**の余りを求める

〈図1-22〉ヘッダ・ファイルsignal.hに宣言されている関数

関　　数	内　　容
void (*signal (int sig, void (*handler) (int))) (int)	シグナルを受信したときの処理 **handler** を登録する．**handler** が登録されていて，シグナルを受信すると，**handler** が自動的に呼び出される．このとき送られてくるシグナルには次のようなものがある． **SIGABRT** 異常終了 **SIGFPE** 算術エラー発生 **SIGILL** 不当命令の実行などの異常 **SIGINT** 割り込み発生 **SIGSEGV** メモリ・アクセス・エラー **SIGTERM** 終了要求 定義済みの **handler** としてプログラムを終了させる **SIG DFL** とシグナルを無視する **SIG_IGN** が定義されている
int raise (int sig)	シグナル **sig** を送信する．送信に失敗するとゼロでない値が返ってくる

〈図1-23〉ヘッダ・ファイル setjmp.h に宣言されている関数

関　　数	内　　容
int setjmp (jmp_buf ev)	**setjmp** が書かれた現在のアドレス情報を **ev** に格納し，**longjmp** で戻ってくるための準備をする．**ev** は事前に型 **jmp_buf** で宣言しておく必要がある．型 **jmp_buf** は，**setjmp.h** 内の **typedef** で宣言されている． **setjmp** による呼び出しならゼロが返されてくる
void longjmp (jmp_buf ev, int v)	**setjmp** でアドレス情報を格納した **ev** のアドレスへ戻る．つまり，外見的には **ev** を設定した **setjmp** から戻ってきたように見える．ただし，このとき **v** の値を **longjmp** から戻り値として **setjmp** が渡すため，**setjmp** からの戻り値をチェックすることで **setjmp** 本体からの戻りか，**longjmp** からの戻りかがわかる．そのため **v** にはゼロ以外の値を設定する必要がある

の二つの関数 **signal** と **raise** を宣言しています(図1-22 参照)．

⑫ **setjmp.h**(非局所的ジャンプ)

ヘッダ・ファイル **setjmp.h** には，通常の関数呼び出しと戻りとは異なる，設定してアドレスへ直接戻るための関数を提供します(図1-23 参照)．

⑬ **assert.h**，**locale.h**，**errno.h**

最後に残った三つのヘッダについて簡単に述べると，**assert.h** は診断マクロ **assert** を定義しているファイルで，プログラム中に **assert**(任意の式)と書くことで，プログラム実行中，この **assert** を実行するたびストリーム **stderr** に，ソース・ファイル名，行番号，そして指定した任意の式が印字(表示)されます．

locale.h には，**setlocale** というプログラムの実行環境を選定，変更する関数があり，この **setlocale** の内容は処理系により異なります．

errno.h は，グローバル変数 **errno** に設定する定数の名前を定義しているもので，処理系によりその内容は異なります．

● **次回の予告**

今回は，第1回目ということで ANSI の C について，文法で変わったところや，ヘッダ・ファイルの内容について触れましたが，次回は **ctype.h** や **string.h** で宣言されている関数やマクロについて，その詳細な使い方について述べたいと思います．

掲載プログラムの頒布方法について

今月号に掲載したソース・リストを MS-DOS のテキスト・ファイルで提供します．ご希望の方は現金書留で代金3,000円(送料・税込み)と下記の申込書のコピーを添えてトラ技コンピュータ　ディスク・サービス係までお送りください．なお，頒布メディアは5インチ2DDです．

申　込　書

★お手数ですが，ご住所とお名前を2箇所に記入してくださいますようお願いします．

★お申し込みされた日付けをメモしておかれると，問い合わせの際に便利です．→　月　日

▼ディスクの送り先を記入してください．

〒170
東京都豊島区巣鴨 1-14-2
C Q 出版(株)　C&E出版部
トラ技コンピュータ　ディスク・サービス係

〒　－

様

申し込み書　〈トラ技コンピュータ　1989年11月号　ディスク・サービス〉　TC8911

ご住所 〒　－

お名前

コンピューティング・エッセイ 1

今フラクタルがおもしろい

山本　強

今回のタイトルを見て半分ぐらいの人は何をいまさらと思ったに違いない．フラクタルといえばすぐコッホ曲線やシェルピンスキー・ガスケット，そしてMandelbrot集合にJulia集合とどことなく怪しい雰囲気をもつ図形を連想するが，じゃあ何の役に立つのと聞けばせいぜいありもしない山の稜線や樹木を作ってCGの背景に使うというところ．まあ，そう割り切って遊んでみればそれはそれで結構楽しいものである．

ところがこのフラクタルに最近新手の応用があらわれて来た．フラクタル画像圧縮というのがそれ．関係者の話を鵜呑みにすれば人の顔なんかは2Kバイトにまで圧縮できると言うんだからこれは聞き捨てならない．山やシダの葉っぱは見るからにフラクタルしてるからできそうだとは思うが，どうして人の顔がフラクタル・モデルで記述できるのかという点には素朴な疑問が残る．本当なら大変なことなので，あわてて資料収集に取りかかった．

そしてわかったことは，この圧縮法は決っして昨日今日の話ではなく1984年ぐらいからぼつぼつ文献が出ていること，そしてキーワードはIterated Function System (IFS)という一種のマルコフ過程で，それを反復して生成されるパターン(アトラクタと呼ぶ)を利用するということである．

つまり目的画像があってそれをアトラクタとするようなIFSのパラメータを逆問題として求めるわけである．一組のIFSパラメータは高々24個の実数であり，それで木の葉程度は書けてしまう．普通の画像で1Mバイト程度の画像が2Kバイト程度まで落ちるそうである．

もっとも圧縮-再生によって原画像はかなりのひずみを受け，写真がまるで油絵になって返ってくる．しかも圧縮は対話的に人手を介して行われるというから，だいぶ話は色褪せてきた．しかし話はこれから始まるのです．

今年の6月にボストンで開かれた とある学会で，この圧縮法の提唱者であるジョージア工科大学のM. Barnsley教授の話を聞く機会があり，最近の動向に触れることができた．最近の圧縮技術の紹介があって，先生自らの御真影の使用前後が示された．

わずか2Kバイトに圧縮されたデータから再生された画像は，原画像として一見して差がわからない．それで驚いてはいけない．彼氏曰く，すでにフラクタル圧縮のエンコーダ/デコーダのハードウェアが開発されており，それによれば一般電話回線で毎秒30コマのモノクロ動画像が送れるというのである．

私はここではたと困ってしまった．というのは，この手の電話回線で動画像が送れる「画期的装置」といった話は永久機関みたいなもので ときどき業界紙を賑わすけどいつのまにか消えていくものだから．でもBarnsley先生というのは立派な著書もあり論文もいっぱい書いてる偉い先生みたいだし(この世界，論文の数の多い人＝偉い人)…ならば 論文を読むなり本人に聞くなりすればいいと言われるかもしれない．

でも，論文に書いてあるのは先に書いた油絵と木の葉の書き方だけ．どこにも顔の写真の圧縮法は書いてない．当然みんなBarnsley先生に質問するわけだけど，先生は肝心なところは一言"No comment"．しまいには講演会がほとんど喧嘩の様相を帯びてきた．結局信ずる者だけが救われるというところか．これは一度実物を見てみたいと思いつついったん帰国．

そして8月．まったく別の学会でまたボストン．そしてまたBarnsley先生．今度は大学ではなくIterated System Inc.という会社名でSparcStation-1の上でムービのデモを見せてくれた．全部ソフトでやってるから毎秒22コマしか再生できないんだそうだ．本当ならそれで十分ではないか．

突っ込んで質問したところ，500倍の圧縮は動画の場合でかつ誤差をかなり許す場合だそうで，1枚物だと100倍以下になるとのこと．でも圧縮は完全自動になったという．ただしハードウェア装置は来年にならないと出荷できないとのこと．もしこれを売りたいのならば ここに連絡をくれ，とビジネス・コンタクト先が書いてあるリーフレットまで貰ってしまった．

結局，その雰囲気に乗せられて先生の本を一冊買ってその場を引き上げた．考えたら，先生は本屋のブースでデモをして そこで本を即売するという商売上手．わたしもそれに乗せられた口なんだろうか．結局，今回も半信半疑で帰国する事となった．

困ったときのネットワークだのみとばかりに，帰ってからしばらくの間junetに流れてくるアメリカ国内のニュース(**comp.graphics**)を注意して眺めていたら，やっぱり興味を持ったのは私だけではなかったようで，しばらくそれに関する議論が続いていた．これがまた賛否両論で，単純に信用する人と相手にしない人が半々というところか．わかったことはどうもAT & Tが絡んでいるらしいということ．これはAT & T Pixel Machineという並列コンピュータがあって，その上でもフラクタル画像圧縮のデモをやっていたというところから来ているらしい．いずれにせよ，来年の前半には話が本当かどうかわかるわけだから楽しみである．

ところで乗せられて買ってきた本であるが，Fractals Everywhere (Academic Press)というタイトルで結構おもしろい．先生の講義のテキストなのだが，まえがきにこんなことが書いてある．

「もしあなたが幼い頃にもっていた素朴な目を失いたくなかったらこの本を読まないほうがよい．この本を読んでしまったら，世の中の全てのものはきわめてわずかのパラメータで記述できてしまうことがわかってしまいもうもとには戻れなくなってしまう．」

なかなかの自信である．残念ながら私はまだその域には達していない．

初心者ならPascalをおすすめします

中野 正次

Pascalとの出合い

電子回路の設計が専門だった筆者が，ソフトウェアに手を出さなければならなくなったきっかけは，コンピュータのハードウェアをテストするためのプログラムが必要になったからでした．それも，最初はソフトウェアの専門家に頼んでいたのですが，適切なものができてこなかったため仕方なしに始めたのです．

そのときの状況は，テスト中のCPUでアセンブラやコンパイラを走らせることができないため，ほとんど直接に機械語でメモリ番地を計算しながらプログラムを書いていました．

しかし，それでは効率が悪すぎるというので，テスト中のハードウェアでも実行できて，スピードも速い簡易型のアセンブラをそのテスト・プログラム専用に作りました．

というわけで，最初はハードウェアに極めて近いレベルのソフトウェアを扱っていました．その後，NC(数値制御工作機械)関係の会社に移って，サーボ制御のハード/ソフトの開発を手がけました．このときもソフトウェアは極めてハードに近いもので，マクロ・アセンブラを使っての作業でした．

そんな中でシステムの大型化が進み，NCプログラムに基づいて加工を連続的に行うことのできるシステムを設計することになりました．

当然，アセンブラでは手に負えないため，コンパイラを使うことになりましたが，マイコン・レベルで制御用に使えるコンパイラの経験は全くなく，とりあえずCP/M80上で走るホワイトスミス社のCコンパイラとPascal/MT＋(当時はMTコーポレーションが供給)を入手して検討してみました．

ところが残念なことに，Pascalのほうはなぜか暴走してしまう現象が起こり，Cコンパイラで計画を進めていくことになりました．

もちろん，Cのコーディングは初めてなので四苦八苦……．同時に何とかPascalを使いたいと代理店に問い合わせたりしていたところへ，CP/M80の中にバグがあるという情報があり(現在でもこのバグはあります)，これを訂正することでPascal/MT＋が完全に走るようになりました．

そこで，Cでコーディング中だったものをPascalに書き直したところ，一ケ月ほどで出荷できました．このソフトは，データ・ファイル中のNC言語を解読し，エラー・チェックを行い，その内容をモデムを通してサーボ用コンピュータに送るもので，データ・ファイル20個を指定しておいて連続運転できるものです．

これが，Z80SIOのモード・セットや通信制御を含めて，すべてPascal/MT＋で書いて153行に収まり，現在も問題なく稼動しています．

このとき，さりげなく使った

```
ARRAY［1..20］OF TEXT；
```

というファイルの配列が，すんなりとコンパイルされて実行できたことが，今でも強く印象に残っています．

それ以降，CP/M80のソフトはすべてMacro80とPascal/MT＋R5.0を使って作ってきました[1]．

また，Pascalの基本機能は移植性がよいので，ヒューレット・パッカード社の9000シリーズ(CPUは68010)上で使用するインタプリタのデバッグをPC8001上でPascal/MT＋でコンパイルして行ったこともありました．

デバッグが完了したソース・プログラムは，そのままではHP9000シリーズで読めないので，フォーマット変換が必要でしたが，これもPascal/MT＋を使って作り，実機上ではわずかの変更のみで完成させることができました．

現在は，Pascal/MT＋は販売されていないようで，CP/MでもMS-DOSでもTurbo Pascalが主流になっているようです．しかし，Pascal/MT＋用のソース・コードはわずかな変更でTurbo Pascal上でもコンパイルできます．

というわけで，筆者にとっては現在に至るまでPascalだからできなかったということは一度もなく，実行スピードの極限を求める用途以外なら，すべてPascalを使うことに決めています．

CやBasicもユーザからの要求があって使わなければならないことがありますが，移植性の悪さ(MS-DOS上でも互換性がない)とスピードではアセンブラにかなわないという点で，自分から使ったり，人に薦めたりすることはありません[2]．

ソフトウェアの種類

多くの市販ソフトはC言語かアセンブラで書かれています．アセンブラは別として，C言語が使われてい

るのはC言語が優れているからでしょうか？　ここで重要なことは“市販ソフト”という点です．市販ソフトは多くのユーザの様々な使い方をあらかじめ考慮しています．

また，使い方を誤ったり，システム自身にバグがあったり(市販されるソフトに本来はバグがあってはならないが，実際にはバグはある)したときに，確実にエラー状態を検出したり表示したりしなければなりません．つまり，ユーザのレベルもピンからキリまで使えるように，メーカには問い合わせが来ないように作られているわけです．また，OSの標準的な機能だけを使っていたのではグラフィックやマウスを使えないので，ROM内のプログラムを直接実行したり，I/Oポートの読み書きを行ったりします．C言語はこのような機械語に近いレベルの記述が比較的簡単なので，市販ソフトの開発に利用されているのです．

しかし，よく考えてみると，ソフトの中にはユーザがたった1人だけ(または1社だけ)ということもあります．また，特定の使い方しかないもの(OSから起動すると，決まった作業をして単にOSにもどるような)もあるわけです．場合によっては，作成者だけがユーザでもあるということも起こります．

さらに，完成したら一度だけしか使わないというプログラムもあり得ます(例えばパズルを解くプログラムなど)．プログラムを汎用的に作ったり，画面をカラフルにしたりするのは，市販品だから必要になる(市場での競争のため，重要でないことに大きな労力を費している)ことであって，上記のような場合にはほとんど考える必要がありません．

また，ソフトを作る人が必ずしもソフトの専門家であるとは限らず，一生に一度しかソフトを作らない人がいてもよいわけです．

“それならBasicを使えばよいではないか”といわれそうですが，論理型，順序型，**RECORD**型などのデータが使えず，新しいデータの定義もできず，**GOTO**や**GOSUB**でどこへでもスッ飛んでしまうBasicでは，Pascalの何倍もの苦労を余儀なくされることになります(Basicにも構造化Basicというものがあり，この場合はデータについてもプログラムの流れについても極めてPascalに近いものになっています)．

PC9801シリーズをはじめ，多くのパソコンにBasicが付属して売られているので，とりあえずBasicから入門する人が必然的に多くなるわけですが，少し手の混んだアルゴリズムが必要であったり，大きな可変長データを扱ったりした場合は，手に負えなくなってほかの言語システムを求めることになります．

もちろん，内蔵Basicから抜け出してOSの上で走る言語をさがしながら，結局OS上のBasicにたどりついてしまうケースもあります(むしろ多い？)が，これはおそらく最悪のケースです．

幸いBasicがいやだと思う人は，C言語を選ぶことになります．それはC言語の解説書が最も多いからでもあります．PDS(Public Domain Software：公開ソフトウェア)のユーティリティなどもC言語のソースで出てくるものが多いようです．しかし，Cの解説書が多いことと，Cが優れていること，またCのユーザが多いこととはすべて別問題なのです．

Cの解説書が多いのは，解説書を書く人が市販ソフトやシステム・ソフトの専門家だからです．この種のソフトを書く人は，C言語を使わないわけにはいきません．そして，C言語を使えば特にほかの言語も使う必要はないわけです．

一方，市販ソフトを作る立場にない人は，一般的に言語の解説書を書いたりもしないので，潜在的には多いPascalのユーザも表に出る機会はCに比べてずっと少ないことになります．それともうひとつ，非常に重要な要素があります．それはPascalはあまり解説を必要としないということです(？！　この原稿はPascalの解説だった…か)．

◆ 参考文献 ◆

(1) 中野正次；実戦マクロ・アセンブラ活用法，CQ出版社．
(2) 中野正次；デバッガ/シミュレータの実際，インターフェース，1987年9月号，pp. 256～289．

Keyword

Turbo Pascal：米国Borland International, Inc. が1983年にCP/M80上で動作するコンパイラとして発表した．インタプリタ感覚で使えることと超低価格が受けて爆発的に使われるようになった．現在はMS-DOS上(IBM PCとPC9801)で動作するものが販売されている．今年になって，㈱ボーランドジャパンが設立されたが，販売は㈱マイクロソフトウェアアソシエイツ(☎03-486-1411)が行っている．Turbo Pascal 5.0は29,800円．

コンパイラ：普通の文章に近い言葉で記述できるのが高級言語で，これを機械語に変換するための翻訳プログラムのことをコンパイラという．変換前のプログラムをソース，変換後の機械語プログラムをオブジェクトともいう．

インタプリタ：コンパイラと同様に高級言語を機械語に変換する翻訳プログラムであるが，コンパイラのようにオブジェクトを作らずに，プログラムを逐次解釈しながら即実行する．したがって，実行速度はコンパイラよりも遅いことが多い．

第1回 システム開発技術者のためのPascalプログラミングの手解き

Turbo Pascal入門

中野 正次
なかの・まさつぐ

Pascalについては多くの解説を必要としないので，本連載ではMS-DOSが直接サポートしていないパソコン上の機能や，自作のハードウェア(オプション・ボードなど)を使用する方法，また8086系のターゲット・システム(パソコン以外のCPUボードなど)のROM上で走るプログラムを作る方法，デバッグする方法をPascalを使って解説することに主眼を置きます．

したがって，内容の多くはほかの言語に置き換えても利用価値があると思われますが，Pascalの記述能力は極めて高いので，簡単に置き換えできるとは保証できません．当然，Pascalで記述するのが最もふさわしいという結論が予想されます．

とはいっても，これまでPascalには全く縁がなかった読者を門前払いにするつもりはありません．筆者としては1人でも多くのプログラマに，またはプログラマの卵にPascalを知って良かった，Pascalを使って良かったと感じていただきたいので，簡単なPascal入門の項を設けます(とにかく，言語というものは使ってみなければわかりません．逆に，使ってみればわかります)．

Pascal入門については一般的なPascalですが，より細かい部分については現在MS-DOSの標準的なPascalコンパイラであるTurbo Pascal(BORLAND社)を使用して解説します．このコンパイラは現在バージョンV5.0となっており，IBM PCとPC9801以外では多くの機能を活かすことができません．

これに対してV3.0ではほとんどのMS-DOSマシン上で問題なく使えます．この点以外にもV3.0のメリットは数多くあり，V5.0は純粋にV3.0からのグレードアップ版ではありません．この両バージョンの差異については後でも触れますが，V3.0の販売継続を望む声もかなり聞かれます．筆者も特に教育用にV3.0(できれば改良版のV3.n)の継続を強く求めたいと思っています．

このような背景のもとで，V5.0でしか動作しないプログラムを使っての解説はあまり適切でないと考え，サンプルとして扱うものはできるだけ両バージョンで動作可能なものを選び，これが不可能な場合は各バージョンでどの部分を変えればよいか明示する形で進めて行くことにします．

また，V3.0とV5.0の一方でしか使えない書き方や機能もあり，これらについては別の項で各バージョンの特徴として扱います．ところで，V4.0はどうなっているんでしょうか．そうなんです．Turbo PascalにはV3.0より古いものもV4.0もあります．しかし，V3.0以前のものはV3.0で，V4.0はV5.0で代表できる程度の差しかありません．

すなわち，V3.0以前とV4.0以降では同じTurbo Pascalという名前で呼ぶには不都合と思われるほど大きな変更が行われています．この点についても後で細かく説明する予定ですが，V4.0以降で大きく変わった点を簡単に紹介すると，

① 64Kバイト以上の大きな実行コードが作れるようになった．したがって，実行ファイルは.**COM**から.**EXE**に変わった．

② ソース・プログラムの分割はV3.0以前では**Include**形式しかなかったのが，**Uses**形式がとれるようになり，分割されたものを別個にコンパイルできるようになった．これによって，デバッグ中のコンパイル時間を短縮できるようになった．

③ アセンブラとのリンクが簡単にできるようになり，アセンブラの中からPascal中のグローバル変数や手続きに直接アクセスできるようになった．

④ 8087(80287)を使うためのコンパイラが1本化された．

⑤ 実行時デバッグ機能が追加された(V5.0から)．

これらのうち，ソース・プログラム作成上で最も重要なのは②の分割コンパイル機能です．Turbo Pascalではこの分割単位を**Unit**と呼び，**Uses**と共に本来のPascalにはなかった概念として導入されています．

また，プログラムを書くことに直接関係はありませんが，コンパクトなコンパイラのイメージはなくなりました．逆に，生成される実行コードはV3.0では最小10658バイトであったのがV5.0では1344バイトと大幅に

小さくなりました．この点はROM化して組み込むプログラムには大きなメリットです．

Turbo Pascalの基本操作

Turbo Pascalはコンパイラなので，インタプリタ形式のBasicと違いプログラムを書いてもコンパイル作業を行わないと実行できません．ただし，Turbo Pascalではエディタを内蔵した形で作られているので，インタプリタと同じ感覚で操作できるようになっています．

Turbo Pascalには多くのサンプル・プログラムやユーティリティが付属していますが，Pascalの基本機能を試してみるにはごく一部のファイルのみで充分です．それらは，Turbo Pascal V 3.0では**TURBO.COM**と**TURBO.MSG**，V 5.0では**TURBO.EXE**，**TURBO.OV1**，**TURBO.TPL**，**TURBO.HLP**です．

これらのうち，**.MSG**と**.HLP**はメッセージ・ファイルですからなくても動作はしますが，使い慣れないうちは入れておいたほうが何かと助かります．

V 3.0では，**.COM**と**.MSG**の二つの合計で40Kバイト足らずですから省略してもメリットはありませんが，V 5.0では四本のプログラムの合計は約490Kバイト，このうち約270Kが**.HLP**ですから若干考えたくなります．といっても，フロッピの2HDタイプなら半分以下ですから重荷になるほどでもないでしょうが，2DDでは明らかに苦しい値です．

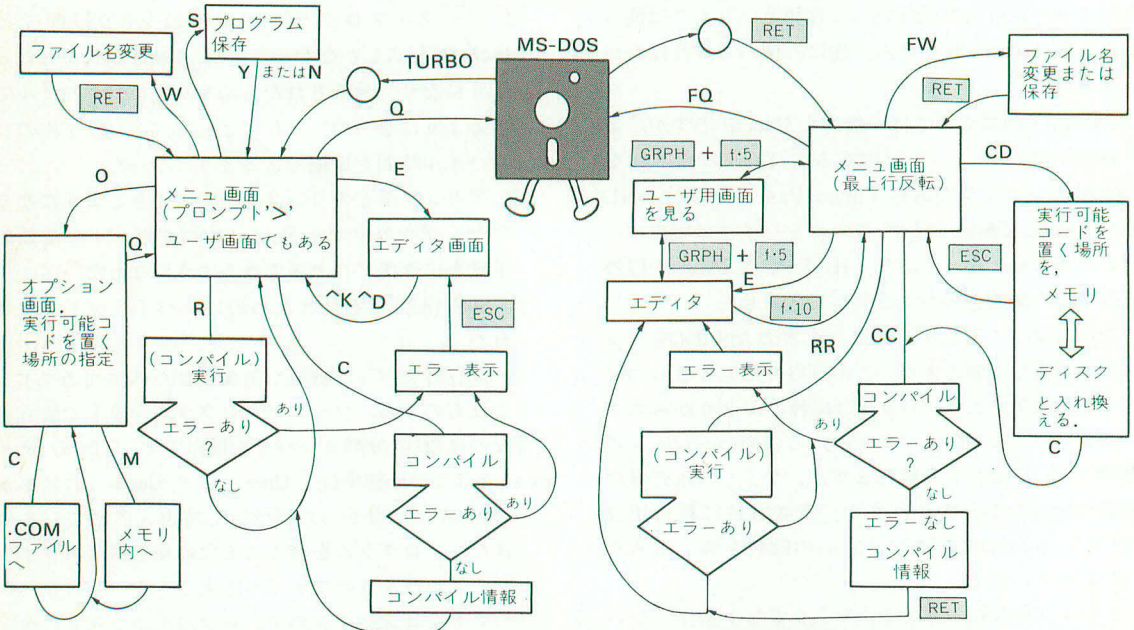
Turbo Pascalの起動は，上記のファイルをカレント・パスに置いておいて，**TURBO**↵で行います．起動後の操作は，画面にしたがってキーを1度押し(V 3.0ではYかN，V 5.0では何でもよい)，メニュ画面に入ります．

その後の操作は**図1-1**に示すようになっています．両バージョンでキーの指定はかなり違っていますが，大きな流れとしてはほとんど同じです．注意を要するのは，V 5.0ではコンパイルされたユーザ・プログラムを実行し終えると，エディタ画面に戻り，ユーザ用画面に出力したものが見えなくなることです．図中にあるように，これを見るには GRPH +f・5 を押します．もう一度押せば，エディタやメニュに戻れます．

ソース・プログラムを書くにはエディタを使うわけですが，V 3.0では特殊キーがサポートされていません(**TINST**でインストールすれば使えますが)．そこで，最低限カーソル移動の CTRL +**A**，**S**，**D**，**F**，**E**，**X**と1文字消去の CTRL +**G**，1行消去の CTRL +**Y**，エディタ終了の CTRL +**KD**程度を憶えておかなければなりません．V 5.0では特殊キーの意味通りの動作をしてくれるのと，HELP キーで操作の説明が得られるので何も憶える必要はありません(ただし，**TURBO.HLP**が見つかるとき)．

そこで，エディタが動作するようなら，Turbo Pascalのシステムを調べるために，次のような最も簡単なプログラムを書いてみましょう．

〈図1-1〉Turbo Pascalの基本操作



(a) Turbo Pascal V3.0の操作．
使用ファイルTURBO. COM，(TURBO. MSG)

(b) Turbo Pascal V5.0の操作．
使用ファイルTURBO. EXE, TURBO. OV1, TURBO. TPL, (TURBO. HLP)

```
PROGRAM v1;
BEGIN
WRITE ('コンニチワ');
READLN
END.
```

これをコンパイルし，実行すると画面上に「コンニチワ」が表示されます．エラーが出るとすれば多くの場合ミス・タイプですが，Turbo Pascal のファイルに欠陥がある場合も全くないわけではありません．エラーがなければ「コンニチワ」が表示されたあと，RET キーを押し，Turbo Pascal へ戻ります．ここまでうまく動作すれば，Turbo Pascal システムは OK と考えられます．後はどんどんプログラムを書きさえすればよいのです．

Pascal の特徴

● Pascal の文法

Pascal は，数多くの言語の中でも最も厳格な文法を規定しているものに属します．変数はすべて型(変数の種類)を指定した宣言を行った後でしか使えませんし，定数も値を与えてから使わなければなりません．

手続き(＝サブルーチン)や関数も定義してから呼び出すようになっています．同じファイル内にあっても，定義より前に呼び出しが来ればエラーになります．この点は C や Basic とは全く違い，「このガンコモノめが」とどなりたくなる方もいるかも知れません．しかし，このことはよく考えてみると，プログラムを先頭から読んで行くと，性質(変数か手続きか，その型や値など)を定義していない単語が使われることがあり得ないということを意味しています．

プログラムを書いているときには，書きやすさが重要ですが，書き終わったら…読みやすさのほうが重要になってくるのです．たとえ 1 回だけしか使わないプログラムであっても，苦労して考えた解決法を書いたものであれば 何かの機会に同じ考え方が利用できるかも知れないし，他人に説明する必要が生じることも考えられます．このようなとき，頭から一度読めばすべてが理解できる(はずです．記憶力の問題は別ですが)という文法は何とありがたいことでしょうか．

● データの型

また，Pascal では論理型(**Boolean**)や集合型(**SET of**)という型が用意されていますが，さらにユーザがその都度新しい型を作り，その型に名前をつけることができます．また，順序型を新しく定義する場合，その値をユーザがつけた名前で表すことが可能なのです．**表1-1** に Turbo Pascal のデータの型を示します．このような機能も書きやすさ以上に読みやすさに役立っています．ただし，Turbo Pascal では残念なことに，ユーザ定義の値を持つ変数をコンソールに書き出し(**WRITE**)たり，コンソールから読み込ん(**READ**)だりすることはできません．

いくつかのデータをまとめてひとつの変数として扱うこともできます．このうち，データの型が同じものをまとめた場合は配列(**Array**)と呼び，Basic でも使われています．ただし，配列の要素になる型には制限がなく，配列型や集合型，さらに後述のレコード型(**Record**)も配列の要素になり得ます．

したがって，配列の次元数には制限がありません．配列変数の中の特定の要素を指定するには，何番目のデータかを示す添字を使いますが，ここにもユーザ定義の値を持つ順序型が使えます．

▶ レコード型データ

データの型が違うものをまとめた場合は，レコード型と呼びます．この場合，レコードの要素は個々に(そのレコード内でのみ有効な)名前をつけておき，レコード全体の名前と要素の名前を . で区切って指定します．

レコードの要素になる型にも制限はなく，配列やレコードも要素になり得ます．このように多くの関連性のあるデータをまとめてひとつのデータとして処理できることは，大局的な考え方をそのまま書き表すのに重要な役割を果たします．

定数は式の型から自動的に型が決まってしまうものと，型を(変数と同じように)指定して定義するものとがあり，ファイル型を除くすべての型の定数が作れます．また，この種の型つき定数は初期値が設定される点以外は基本的に変数と同じなので，別の値を代入し直すことも一応可能です．ただし，この場合は定数としての役目はなくなります．

▶ ファイル型データ

ファイル型は，変数の値がファイルの内容を表すという性格のものではなく，OS とユーザ・プログラムの間でやり取りされるファイルの状態を保持するものです．したがって，ファイル型の変数は代入の対象として許されていません．

しかし，実際にはファイル型を含むレコード型の代入は可能であり，ユーザの責任において変わった使い方も考えられるかも知れません．

また，ファイル型は比較や演算の対象にもならず，ほかの変数と比べて全く違う決められた取り扱いをしなければなりません．ファイルの扱いについては，操作例のところで詳しく述べることにします．

▶ ポインタ型データ

ポインタ型はプログラム実行中に，新たなデータ保存場所を作りだすために使われます．ポインタ型は一般の演算の対象にはならず，代入と等しいかどうかの

〈表1-1〉
Turbo Pascal の
データ型(1)

型　名	内　容	バイト数	範　囲　な　ど	備　考
CHAR	文字コード	1	#0～#255(#は文字コード化演算子)	
BYTE	正整数	1	0～255	
SHORTINT	整　数	1	−128～127	V 5.0
INTEGER	整　数	2	−32768～32767	
WORD	正整数	2	0～65535	V 5.0
LONGINT	整　数	4	−2147483648～2147483647	V 5.0
REAL	実　数	6	有効桁数≒11	V 5.0
SINGLE	実　数	4	有効桁数≒7	
DOUBLE	実　数	8	有効桁数≒15	V 5.0
EXTENDED	実　数	10	有効桁数≒19	V 5.0
COMP	整　数	8	$-2^{63}+1$～$2^{63}-1$	V 5.0
STRING [n]	文字列	n+1	最大長さnの文字列．nの最大値は255	
BOOLEAN	論理値	1	値は **TRUE** と **FALSE** の2種だけ	
FILE	ファイル・コントロール用のデータ	128	内部構造については一般に知る必要はない	
TEXT	FILE の特別な型	256	文字列ファイルで **READ** や **WRITE** で自動変換ができるもの	

〈表1-1〉 Turbo Pascal のデータ型(2)

型　名	内　容	書　式　な　ど
ARRAY	配列（同じ型のデータをまとめたもの）	**ARRAY** [**n1**..**n2**] **OF** 任意の型　（任意の型には **ARRAY** や **RECORD** も含む） **ARRAY** [**n1**..**n2**] **OF ARRAY** [**m1**..**m2**] **OF** □ は **ARRAY** [**n1**..**n2**,**m1**..**m2**] **OF** □ と書いてもよい
RECORD	レコード（異なる型のデータをまとめたもの）	**RECORD** 名前 : 任意の型 ; 名前 : □ ; 名前 : □ **END**　いくつでも書ける ↑この名前をフィールド名と呼ぶ
(**ordinal**) 型にはユーザが名前をつける	順序型（ユーザが値を指定するもの）	(**right**,**left**) や (**apple**,**banana**,**cherry**) など． **30**..**70** や **'A'**..**'Z'** のように整数や文字型の部分範囲を指定することもできる．
SET	集合型	**SET OF** 順序型のうち値が256種類以下の型名 バイト数は最大32．ビット単位で元の存否を保持する．
(**pointer**)	ポインタ型（データなどのアドレス）	^ 任意の型名　^がポインタ型を作り出す． V 5.0 では **pointer** という指すべきものを指定しないポインタ型が使える

比較しかできません．Turbo Pascal では，プログラムの実行文が扱えるデータ(グローバル・データと呼ぶ)は64Kバイトまでなので，大量のデータを扱うにはポインタによるしかありません．

また，ポインタを利用すると，物理的なデータの位置(メモリ内のアドレス)と無関係な順序付けをすることができ，データの並べ替え(いわゆるソート)に威力を発揮します．ポインタを使って新しくデータを作る場所をヒープ領域と呼び，ユーザ領域の残り全部(プログラム，データ，スタック以外)を割り当てることができます．

● プログラムの構造

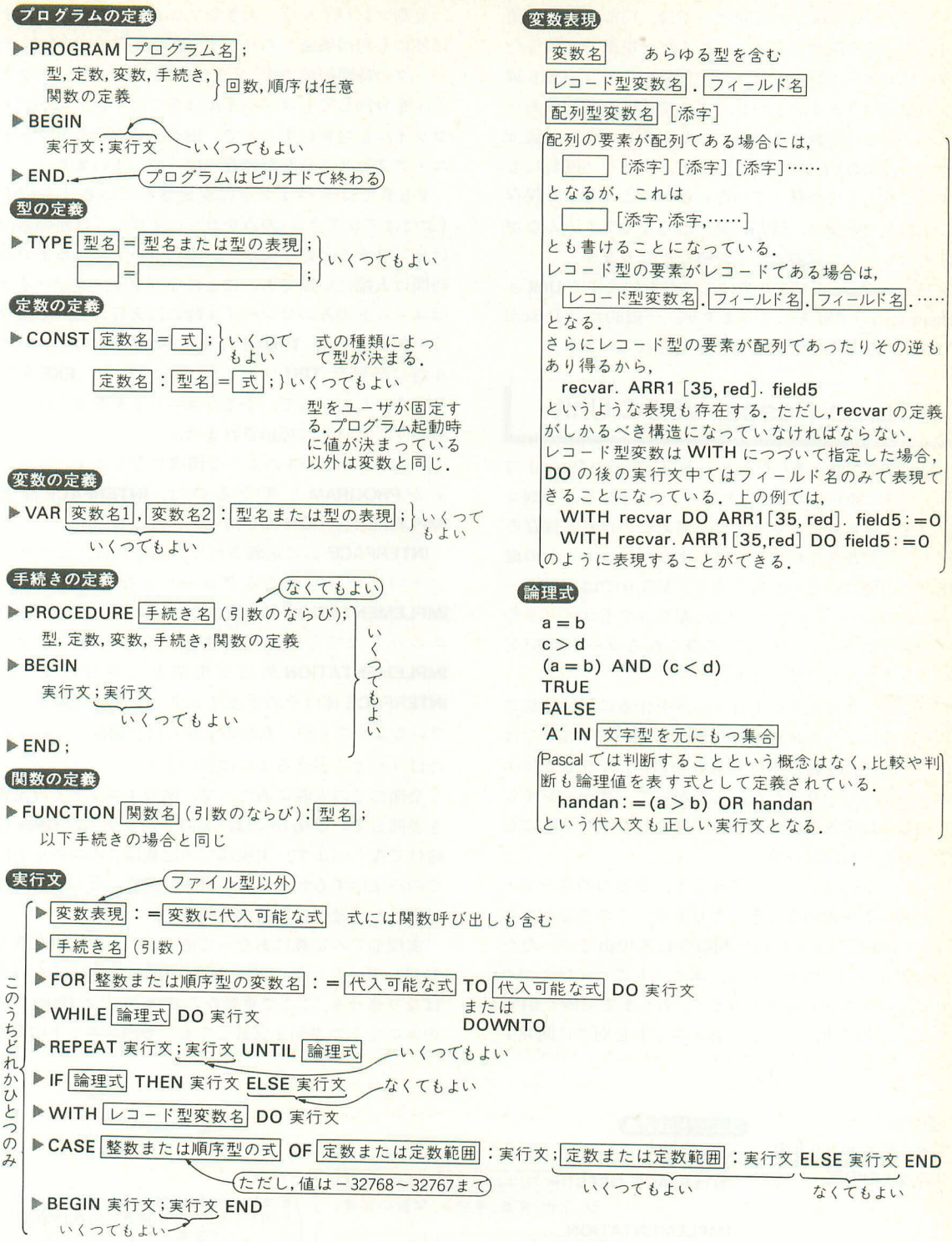
Pascal のプログラムは **PROGRAM** で始まり，**END.** で終わります．これは，ほとんどすべての Pascal で共通となっていますが，内部の構成は標準の Pascal と Turbo Pascal では少し違い，Turbo Pascal のほうが制限が緩くなっています．Turbo Pascal ではプログラムの構造は**図1-2**のように定義されています(厳密には，一部省略してあります)．

この中で最も重要なことは，**PROGRAM** から **END.** までの内容と **PROCEDURE** から **END;** までの内容が全く同じという点です．**FUNCTION** については，関数が呼び出し側へ返す値の型を指定する以外は，これまた全く同じです(要は手続きとは値を返さない関数と考えればよいわけです)．

すなわち，**PROGRAM** は1度しか使われませんが，**PROCEDURE** や **FUNCTION** は，その内部に変数や定数，手続きや関数を定義できるのです．これはデータの型で配列やレコードの要素が配列やレコードでもあり得るのと同じように，無制限に多重化できます．また，手続きや関数の中で，その手続き(関数)自身を呼び出すこと(リカーシブ＝再帰的な呼び出し)も可能です．もちろんこの場合，何らかの方法で再帰呼び出しを止める工作をしておかなければなりませんが，使い方によっては解法を明確に表す手段にもなります．

Pascal という言語は，様々な局面で無制限に多重化できるように作られています．このことは，大局的に

〈図1-2〉 Turbo Pascal のプログラムの定義



考えるときも各局部について考えるときも，さらに細部について考えるときも同程度の複雑さの中で作業ができることを意味しています．しかし，単に多重化できるだけでは不充分です．Pascal では多重化に際して混乱を避けるように重要なからくりが作られています．

それは，手続きや関数の中(すなわち **PROCEDURE** または **FUNCTION** から **END** までの間)から外は見えますが外から中は見えない，というハーフ・ミラーのようなからくりです．

この機能によって，関数や手続き内でのみ使う名前は，その外側で定義されているものとダブっても何ら問題は起こりません．ただし，ダブった名前について

は外側のものは見えなくなります．

プログラム作成上の要領としては，局部的にしか使用しない値や関数などは，できるだけ局部で定義したほうがエラーの発生を防ぎ，実行時のメモリ消費も減ることになります．しかし，ここで重要な問題があります．それは関数や手続きの中で定義した変数の値は，一度その関数(手続き)から出ると，再びその関数にもどっても，もはや残っていないということです．保存しておきたい値は，ほかに見せたくなくてもみんなが見えるところに定義せざるを得なくなります．

Turbo Pascal V5.0 ではこの点を解決する **Unit** と **Uses** を用いて解決していますが，一般的には Pascal では見えなくなると消えると考えられています．

Turbo Pascal V5.0 の Unit

V4.0 から導入されたユニットは，Turbo Pascal のコンパイル単位(単位だからユニット)となり，分割コンパイルを可能にすると共に，見えないけれど保存されている変数を作れるようにしました．ユニットの機能はこの他にいくつかあります．V3.0 では実行コードが 64 K バイトまででしたが，V5.0 でもユニットやプログラムをコンパイルして得られるコードは 64 K バイトまでとなっています．

そこで，より大きな実行コードを作るには，分割コンパイルによらなければなりません．また，V3.0 ではオーバレイが手続き単位であったのが，V5.0 ではユニット単位になりました．したがって，オーバレイしたい部分は大きさにかかわらず独立したユニットにしておかなければなりません，

ユニットはコンパイルされると，ひとつのコード・セグメントを占めることになります．このことがユニットが 64 K バイト以内に制限される理由です．ただし，グローバルなデータは，ユニットごとにひとつのセグメントを占めるのではなく，あくまで全体で 64 K バイト以内です．この点は各ユニットを別々に開発するときには要注意です．

分割コンパイルは，大きなプログラムやオーバレイ以外にも利用価値があります．それは，コンパイル-デバッグの時間短縮です．V3.0 ではたとえソース・ファイルを分割してもコンパイルは全てのインクルード・ファイルを対象にするので，細かく分割するとファイル・アクセスの時間だけ遅くなってしまいます．

V5.0 ではデバッグ中に変更されているユニット(またはプログラム)のみをコンパイルし，ほかのものはリンクするだけですから，実行コードを作るまでの時間は大幅に短縮できることになります．コンパイラはユニットのみのコンパイル時には実行コードではなく，中間コード(**.TPU**)を作り，プログラムのコンパイル時に必要な **.TPU** ファイルとリンクして **.EXE** を作ります．したがって，小さなユニットを数多く作っても必ずしも時間は短縮されません．

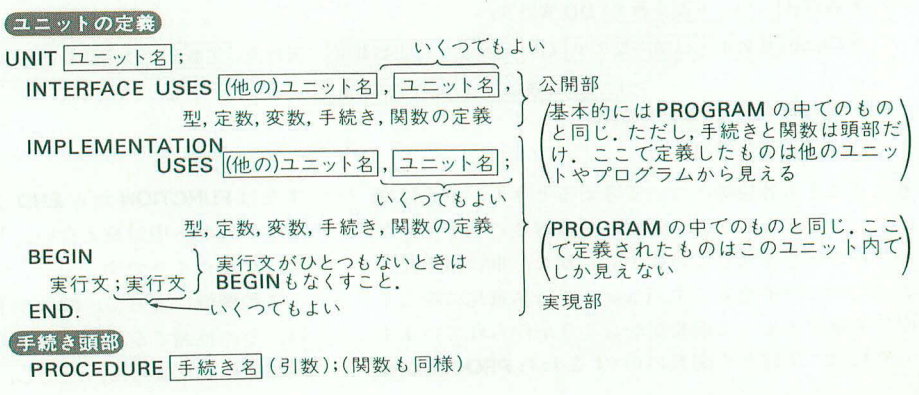
ユニットは図1-3 のような構成になります．ユニットが **PROGRAM** と異なるのは，**INTERFACE** 部と **IMPLEMENTATION** 部を持っていることです．

INTERFACE 部に定義されているものは，このユニットの外からも見えるグローバルなものとなり，**IMPLEMENTATION** 部に定義されているものは，このユニット内でしか参照できないものとなります．**IMPLEMENTATION** 部は実現部とも呼ばれます．**INTERFACE** 部はそのままインターフェース部と呼んでいるようですが，意味の上からは公開部とでもいったほうがよく表せるように思います．

公開部での定義にあたって，他のユニットでの定義を参照している場合には，そのユニット名を **Uses** に続けてならべます．実現部での定義はこのユニット内でのみ使用するデータと手続き，関数，そして公開部に書いた手続きと関数の中味です．

実現部での定義にあたって他のユニットでの定義を参照しているときには，やはり **Uses** で指定しなければなりません．ここで重要なことは，たとえ **Uses** で他のユニットの参照を宣言しても，参照ユニットの公開

〈図1-3〉
Turbo Pascal V5.0 の
ユニットの内部



部に書いてあるもの以外は見ることができません.

このように，何を公開するかを明確に宣言することによって，公開部さえ変更しなければ，これを参照している側のユニットを変更する必要は少ない(少なくとも文法上はない)ことを意味しています.

PROGRAM には公開部はありません．したがって，実現部を宣言する **IMPLEMENTATION** というキーワードもありません(図1-3 参照)．ただし，V 5.0 で **system** ユニット以外のユニットを参照している場合はユニットの実現部と同じように，データの定義に先立って **Uses**…を書かなければなりません.ユニットの実現部にある実行文(必ずしもなくてもよいが)は，**PROGRAM** の **Uses** のあとに指定された順に実行され，**PROGRAM** の中の実行文は最後に実行されることになります.ユニットの中の実行文は，通常，**PROGRAM** から見えなくなる変数などの初期化に使用されます.

V 5.0 では，基本機能は自動的にリンクされる **system** ユニットに収められていますが，V 3.0 では標準であった型や変数，手続きや関数も，V 5.0 ではほかのユニットに分けられたものが多くあります．これは一見 **Uses** 文で指定する手間がかかるとも考えられますが，実は不必要な時に余計な名前の重複の心配をしなくてもすむという大きなメリットを生みます.

system ユニットをはじめ，その他の標準的なユニットはすべて **TURBO.TPL** という特別な中間ファイルになっています．ユーザ定義のユニットは1ファイルに1個だけですが **TURBO.TPL** だけは何個かのユニットをまとめて持っているのです．ユーザ定義のユニットをまとめて～.**TPL** なるファイルを作ってもコンパイラに参照させる方法がありません(作る方法はありますが！).

もうひとつ困ることがあります．それはユニット名はその頭から8文字がファイル名でなければならないということです.ほかの名前がすべて 63 文字まで識別されるのに，ユニット名だけは(ファイル名と共用するため)たったの8文字…これはバランスが悪すぎます.ユニット名に¥を含めることができればサブディレクトリの指定ができ，ファイルの識別能力が上がりますが，これも今のところできません.なお，V 4.0 では{**$U**}コンパイラ指令によってユニット名と異なるファイル名を指定できましたが，V 5.0 では廃止されました.

● Turbo Pascal と Modula-2 の構造

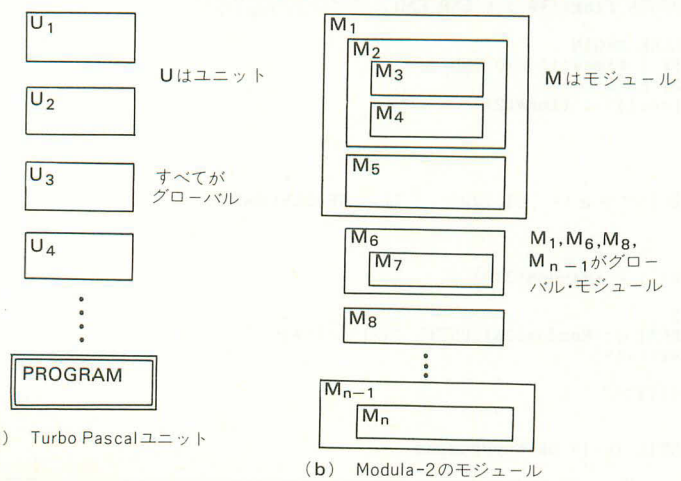
Modula -2 を知っていれば，V 5.0 の **Unit** と Modula-2 の **Module** とが対応していることにはすぐ気付くでしょう．しかし，図1-4 に示すように V 5.0 の **Unit** は多重構造を許していません．すなわち，**Unit** や **PROGRAM** は **Unit** を含むことができません.

これに対して **Module** は **Module** を何重にでも含むことができます．この意味では V 5.0 の **Unit** は「何にでも多重構造を許す」という Pascal 言語の基本思想からはずれています.

Modula-2 はグローバルなモジュールがひとつのファイルになるので，内部にいくつモジュールがあってもよく，数千個のモジュールからなるプログラムも作れますが，Turbo Pascal V 5.0 のほうは，すべてのユニットがグローバルなので数十個程度が実用範囲です.

Modula-2 は優れた言語ですが，現在のところよいコンパイラが作られていません．Turbo Pascal V 5.0 が **Unit** の多重化を許せば，“Modula-3”にもなり得ると思いますが…いかがでしょうか.

〈図1-4〉 Turbo Pascal V 5.0 と Modula-2 の構造



(a) Turbo Pascalユニット　　(b) Modula-2のモジュール

Turbo Pascal を使って周辺回路を操作する例
—モールス練習用符号発生器(乱数型)—

Turbo Pascal の実例として，筆者が2アマ(第2級アマチュア無線技士)の国家試験を受験するために作ったモールス練習用符号発生プログラムを紹介します.

PC9801 にはタイマを使って割り込みを発生する回路や電子ブザーを ON/OFF する回路が内蔵されているので，モールス符号の音を出すために，ハードウェアを追加する必要はありません.

モールス符号を発生させるには，いくつかの問題点が挙げられます.まず，どのような文字を符号化するのかです．国家試験では一応英文の普通文となっていますが，その内容は決まっていません(当然です！).普通文なら英字新聞や英語の教科書から引用すればよさそうですが，こ

〈図1-5〉モールス信号の発生手法

```
A  ‐ ―          (ト ツー)
B  ― ‐ ‐ ‐      (ツー トトト)
C  ― ‐ ― ‐
:
J  ‐ ― ― ―      (アルファベットの中で
:                最も長い符号)
```

(a) モールス符号の通常の表現

例1

```
CASE Ch OF
'A':BEGIN S;L END;        (S はON1, OFF1
'B':BEGIN L;S;S;S END;     L はON3, OFF1
:                          を作り出す手続き)
END;
```

例2

```
CASE Ch OF
'A':BEGIN P(1);P(3) END;
'B':BEGIN P(3);P(1);P(1);P(1) END;
:                          (P は引数の長さのONと
END;                        1のOFF を作る)
```

例3

```
CASE Ch OF
'A':q (1,3,0,0);   (qは四つのパラメータを順に調べ,
'B':q (3,1,1,1);    0でなければ上のP(n)の動作をし,
:                   0なら何もしない. 0で何もしない
END                 のは, 1のOFFが重ならないように
                    するため.)
```

(b) プログラム例

れをファイルに打ち込むのが大変です.

ここで手を抜いてわずかな例文だけを繰り返し練習していると，受信するよりも暗記してしまって困ります．一方，手もとにある英文ファイルはほとんど Pascal のソース・プログラムで, **PROCEDURE** や **INTEGER** がやたらに出てきて，とても普通文とはいえません．

そこで乱数を使ってデタラメな英字のならびを乱数で作った数だけ並べたものを単語とし，これを一定文字数ならべてひとつの文にすることにしました．これなら次の文字を予測することができないので，練習用には効果絶大！だと思いましたが….

2 アマでの問題文は 90 文字程度なので，文字数は 100 文字としました. あまり長くすると, 途中で息が切れてしまいます．慣れないうちは短かめから始めたほうがよいかも知れません．

● トン・ツーの表現方法

つぎに，モールス符号(トンツー　・―)をどのような形で Pascal 言語の中で表現するかという問題があります. まず, 図1-5 のように **CASE** 文を使う方法が考えられます. この例では, すべて **Ch** に代入されている文字コードによって **CASE** の中のひとつの文が実行される形になっています．当然，文字の数だけの実行文がならぶことになります．ここで使用されている手

〈リスト1-1〉タイマを使わないモールス練習用プログラム(V 3.0 用)

```
PROGRAM morexa ; (* Morse generator for T.P.V3 *)
VAR i,j,k,l,m,n,spd : INTEGER ;
  st : STRING[120] ; c : CHAR ;
PROCEDURE send(cc:CHAR) ;
  TYPE mcode = ARRAY['A'..'Z',0..3] OF INTEGER ;
  CONST mcodes : mcode = ((1,3,0,0),(3,1,1,1),(3,1,3,1),(3,1,1,0),
      (1,0,0,0),(1,1,3,1),(3,3,1,0),(1,1,1,1),(1,1,0,0),(1,3,3,3),
      (3,1,3,0),(1,3,1,1),(3,3,0,0),(3,1,0,0),(3,3,3,0),(1,3,3,1),
      (3,3,1,3),(1,3,1,0),(1,1,1,0),(3,0,0,0),(1,1,3,0),(1,1,1,3),
      (1,3,3,0),(3,1,1,3),(3,1,3,3),(3,3,1,1)) ;
    flagt : INTEGER = 0 ;
  VAR i : INTEGER ;
    PROCEDURE buzon ; BEGIN INLINE ($B4/$17/$CD/$18) END ; ブザーON  ソフトウェア割り込み
    PROCEDURE buzoff ; BEGIN INLINE ($B4/$18/$CD/$18) END ; ブザーOFF ソフトウェア割り込み
    PROCEDURE timew(n:INTEGER) ;
      VAR i : INTEGER ;
      PROCEDURE t; VAR cnt : INTEGER ;                        } 手続きtをプログラムの
        BEGIN FOR cnt:=0 TO spd*2000 DO END ;                 } ループとした.
      BEGIN FOR i:=1 TO n DO BEGIN flagt:=0 ; t END END ;     CPUのクロックによっ
    PROCEDURE q(n:INTEGER) ;                                    てスピードが変わる
      BEGIN IF n=0 THEN n:=n ELSE BEGIN
      buzon ; timew(n) ; buzoff ; timew(1) END END ;
  BEGIN IF cc=' ' THEN timew(3) ELSE
    FOR i:=0 TO 3 DO q(mcodes[cc,i]) ; timew(2)
  END ;

BEGIN (* main *)
REPEAT
  WRITE('   送信スピードを指定して下さい  (4..29) ・・ ') ; READLN(spd) ;
UNTIL (spd>3) AND (spd<30) ;
st:='' ; n:=0 ;
REPEAT
  m:=Random(10) ; k:=Random(20) ; k:=Random(300) ;
  IF n+m+1 > 100 THEN n:=0
  ELSE BEGIN n:=n+m+2 ; l:=255 ;
    FOR k:=0 TO m DO BEGIN REPEAT j:=Random(26) UNTIL j<>l ; l:=j ;
      st:=st+chr(j+65) END ; st:=st+' '
     END
UNTIL n=0 ; l:=length(st) ; st[l]:='.' ;
WRITELN ; WRITE(st) ;
k:=1 ;
REPEAT send(st[k]) ; k:=k+1 UNTIL (k=l) OR KeyPressed
END.
```

続き**S**や**L**，**P**や**q**は図中に示してあるような動作をしてくれなければなりません．

三つの例の中では，例3の**q**がすべて四つの整数をわたす手続きになっているので，これをひとつの配列と考え，さらにこれを**A**から**Z**までの26個の配列として定義しておけば，その型(**ARRAY**[**'A'..'Z',0..3**] **OF INTEGER**)の定数としてモールス符号を与えることができます．これがPascalとしては最もスマートな手法と思われます．

● 時間のコントロール

時間の作り方も，プログラムのループをカウントする方法で一応は動作しますが，クロック周波数やCPUチップの違いによってスピードが変わることになり，あまり面白くありません．モールス符号を作るだけならこれでもかまいませんが，測定用などにも応用することを考えて，タイマ割り込みを使うことも考える価値はあります．

タイマ割り込みを簡単に使うには，BIOS ROM内のルーチンを利用します．ROM内ルーチンの利用法については，Turbo Pascalだけでなく，MS-DOSもサポートしていません．したがって，利用法はROM BIOSの内容について解説した文献を参照するしかありません[PC9801テクニカルハンドブック(アスキー)など]．

電子ブザーについても，ASCIIコードの7(Bellコード)をコンソールに**WRITE**することで一定時間鳴らす機能があるだけで，任意の長さのON/OFFを行うにはタイマ同様ROM内ルーチンを利用することになります．

以上の考え方で実際に動作するプログラムは，**リスト1-1**～**リスト1-3**に示すものになりました．このう

〈リスト1-2〉タイマをインライン機械語で使用したモールス練習用プログラム(V 3.0用)

```
PROGRAM morexa ; (* Morse generator for T.P.V3 *)
VAR i,j,k,l,m,n,spd : INTEGER ;
  st : STRING[120] ; c : CHAR ;
PROCEDURE send(cc:CHAR) ; ひとつの文字を送り出す
  TYPE mcode = ARRAY['A'..'Z',0..3] OF INTEGER ;
  CONST mcodes : mcode = ((1,3,0,0),(3,1,1,1),(3,1,3,1),(3,1,1,0),
      (1,0,0,0),(1,1,3,1),(3,3,1,0),(1,1,1,1),(1,1,0,0),(1,3,3,3),
      (3,1,3,0),(1,3,1,1),(3,3,0,0),(3,1,0,0),(3,3,3,0),(1,3,3,1),
      (3,3,1,3),(1,3,1,0),(1,1,1,0),(3,0,0,0),(1,1,3,0),(1,1,1,3),
      (1,3,3,0),(3,1,1,3),(3,1,3,3),(3,3,1,1)) ;
    flagt : INTEGER = 0 ;
  VAR i : INTEGER ;
    PROCEDURE buzon ; BEGIN INLINE ($B4/$17/$CD/$18) END ; ブザーON
    PROCEDURE buzoff ; BEGIN INLINE ($B4/$18/$CD/$18) END ; ブザーOFF
    PROCEDURE timew(n:INTEGER) ;      時間待ちn×SPD×10ms
      VAR i : INTEGER ;
      PROCEDURE t; BEGIN INLINE (   (*
    000*) $8C/$C8/          (*      MOV AX,CS
    002*) $8E/$C0/          (*      MOV ES,AX
    004*) $E8/6/0/          (*      CALL L0
    007*) $FF2E/6/flagt/(*     INC WORD PTR CS:[mmmm]
    008*) $CF/              (*      IRET
    009*) $5B/              (* L0: POP BX
    00A*) $B4/2/            (*      MOV AH,2
    00C*) $8B/$E/spd/       (*      MOV CX,spd  ;time constant unit=10ms
    010*) $CD/$1C/          (*      INT 1CH     ;timer start
    012*) $C033/            (*      XOR AX,AX
    014*) $3B2E/6/flagt/(* L1: CMP AX,CS:[mmmm]
    019*) $74/$F9)          (*      JZ  L1      *)
                  END ;
      BEGIN FOR i:=1 TO n DO BEGIN flagt:=0 ; t END END ;
    PROCEDURE q(n:INTEGER) ;   ひとつの符号を出す. nが1ならトン
      BEGIN IF n=0 THEN n:=n ELSE BEGIN        nが3ならツー
      buzon ; timew(n) ; buzoff ; timew(1) END END ;
  BEGIN IF cc=' ' THEN timew(3) ELSE
    FOR i:=0 TO 3 DO q(mcodes[cc,i]) ; timew(2)
  END ;

BEGIN (* main *)                                  漢字のメッセージが出せる
REPEAT
  WRITE('    送信スピードを指定して下さい   (4..29) ･･ ') ; READLN(spd) ;
UNTIL (spd>3) AND (spd<30) ;
st:='' ; n:=0 ;
REPEAT
  m:=Random(10) ; k:=Random(20) ; k:=Random(300) ;
  IF n+m+1 > 100 THEN n:=0  文字列長は100まで
  ELSE BEGIN n:=n+m+2 ; l:=255 ;                          同じ文字が続かないように
    FOR k:=0 TO m DO BEGIN REPEAT j:=Random(26) UNTIL j<>l ; l:=j ;
      st:=st+chr(j+65) END ; st:=st+' '
    END
UNTIL n=0 ; l:=length(st) ; st[l]:='.' ;  最後の' 'を'.'に変える
WRITELN ; WRITE(st) ;
k:=1 ;
REPEAT send(st[k]) ; k:=k+1 UNTIL (k=l) OR KeyPressed
END.
```

MOV AH, 17H
INT 18H
MOV AH, 18H
INT 18H
の機械語コード

モールス符号を数字として表す方法のひとつ

ここが割り込み処理のアドレス

割り込み処理アドレスをBXに得る

flogtが0でなくなるのを待つ

IF n<>0とすれば，n:=n ELSEは不要

キーを押せば，その文字を打ち終わったところで終了する

ち，**リスト1-1** と**リスト1-2** はV3.0用で，**リスト1-3** はV5.0用です．V3.0用ではソフトウェア割り込みを **INLINE** 機械語で書いていますが，手続き **INTR** も使えます．**リスト1-1** では時間を作るのにプログラムのループ(**FOR** 文)を使い，**リスト1-2** では割り込み待ちを行っています．

リスト1-2 の手続き **t** の機械語ルーチンでは，タイマにタイム・コンスタント(CXに設定＝**spd** の値)をセットしてスタートし，**flagt** が0でなくなるのを待ちます．タイマ回路はタイムアップすると **flagt** をインクリメントします．このとき，セグメント・レジスタDSは保証されていませんから，データ・セグメントにはそのままアクセスできません．ここでは **flagt** を型つき定数としてコード・セグメント内に配置し，CSを使ってアクセスしています．

型つき定数がコード・セグメント内に作られるのはV3.0までで，V4.0以降ではデータ・セグメントになっています．したがって，非常に便利な上記の割り込み処理法はV5.0では使えません．

INLINE の中で，**CALL LO** を使っているのは，割り込み処理ルーチンを飛び越すためと，割り込み処理番地をBXに取り込むためです．**INLINE** 文や **EXTERNAL** で取り込む機械語コードは，作成時にはどの番地に配置されるかわかりません．**CALL** は実行時に番地を知るひとつの方法なのです．

リスト1-3 では **INLINE** 文は使わず，すべてTurbo Pascalの表現法を使っています．V5.0での割り込み処理はオプションの **INTERRUPT** を加えた手続き **ih** で行っています．この形式の手続きは，入口ですべてのレジスタをスタックにセーブ(プッシュ)し，DSをデータ・セグメントを指すように設定してくれます．

したがって，この中からすべてのグローバル変数がアクセスできるのです．セーブしたレジスタは出口で自動的に復元されます．このレジスタのセーブと復元にはかなりの時間がかかります．高速処理が要求されるときは，V3.0のように **INLINE** で処理せざるを得ないでしょう．

さて，実際にこの乱数型モールス練習器で練習して

〈リスト1-3〉機械語なしでタイマを使用したモールス練習用プログラム(V5.0用)

```
PROGRAM morex5 ; (* Morse generator for T.P.V5 *)
USES DOS,CRT ;                                        標準ユニットを使用
VAR i,j,k,l,m,n,spd : INTEGER ;
  st : STRING[120] ; c : CHAR ;
  flagt : INTEGER ;

PROCEDURE send(cc:CHAR) ;
  TYPE mcode = ARRAY['A'..'Z',0..3] OF INTEGER ;
  CONST mcodes : mcode = ((1,3,0,0),(3,1,1,1),(3,1,3,1),(3,1,1,0),
    (1,0,0,0),(1,1,3,1),(3,3,1,0),(1,1,1,1),(1,1,0,0),(1,3,3,3),
    (3,1,3,0),(1,3,1,1),(3,3,0,0),(3,1,0,0),(3,3,3,0),(1,3,3,1),
    (3,3,1,3),(1,3,1,0),(1,1,1,0),(3,0,0,0),(1,1,3,0),(1,1,1,3),
    (1,3,3,0),(3,1,1,3),(3,1,3,3),(3,3,1,1)) ;          モールス符号を数字で表す方法のひとつ

  VAR i : INTEGER ;  regs : REGISTERS ;               REGISTERSはDOSユニット内で定義
  PROCEDURE buzon ;  BEGIN regs.ah:=$17 ; INTR($18,regs) END ;   ブザーON
  PROCEDURE buzoff ; BEGIN regs.ah:=$18 ; INTR($18,regs) END ;   ブザーOFF  INTRもDOSユニット
  PROCEDURE ih ; INTERRUPT ;                          インタラプト処理
    BEGIN INC(flagt) END ;                            中味は単にflagtを+1するだけ
  PROCEDURE timew(n:INTEGER) ;
    VAR i : INTEGER ;
    BEGIN FOR i:=1 TO n DO WITH regs DO BEGIN
      flagt:=0 ; cx:=spd ; ah:=2 ; bx:=OFS(ih) ; es:=SEG(ih) ;   CXがタイマの時間を決める
      INTR($1C,regs) (* Timer start *) ;                         (CX×10)ms
      REPEAT UNTIL flagt>0                                       ES：BXが割り込み処理
    END END ;                                                    アドレスとなる
  PROCEDURE q(n:INTEGER) ;
    BEGIN IF n>0 THEN BEGIN
      buzon ; timew(n) ; buzoff ; timew(1) END END ;
  BEGIN IF cc=' ' THEN timew(3) ELSE
    FOR i:=0 TO 3 DO q(mcodes[cc,i]) ; timew(2)
  END ;
BEGIN (* main *)
REPEAT
  WRITE('    送信スピードを指定して下さい  (4..29) ･･ ') ; READLN(spd) ;
UNTIL (spd>3) AND (spd<30) ;                          RandomizeはV3.0では不要だった
st:='' ; n:=0 ; Randomize ;
REPEAT
  m:=Random(10) ; k:=Random(20) ; k:=Random(300) ;
  IF n+m+1 > 100 THEN n:=0
  ELSE BEGIN n:=n+m+2 ; l:=255 ;
    FOR k:=0 TO m DO BEGIN REPEAT j:=Random(26) UNTIL j<>l ; l:=j ;
      st:=st+chr(j+65) END ; st:=st+' '
    END
UNTIL n=0 ; l:=length(st) ; st[l]:='.' ;
WRITELN ; WRITE(st) ;
k:=1 ;                                                文字列型は配列としても参照できる
REPEAT send(st[k]) ; k:=k+1 UNTIL (k=l) OR KeyPressed  KeyPressedはCRTユニット
END.
```

受験したところ，合格はしましたが“楽勝”ではありませんでした．というのは，普通文では短い符号(E, I, T など)が多く，同じ時間では乱数の場合よりずっと文字数が多いのです．特に，She is his sister (・・・ ・・・・ ・ ・・ ・・・ ・・・・ ・・ ・・・ ・・・ ・・ ・・・ — ・・—・)のような符号は乱数では起こり得ず，実際にはよく使われるという典型的な例で，文の作り方にはまだ一考(多考？)の余地がありそうです．

集合変数の直接ビット出力への応用

集合変数(**SET OF** ～)は，Pascal の極めて特徴的な概念です．集合型変数は，メモリ上にビット対応で記憶され，その1ビットごとに順序型で定義した名前をつけることができます．また，Turbo Pascal では変数を配置する場所をオプションの **ABSOLUTE** 宣言で別の変数と同じアドレス(つまり，2重の名前と型を同じメモリ上のデータにつける)に配置できます．

この機能を使って8個の元をもつ集合型を **BYTE** としても読み出せるようにしておき，入出力にあたっては **BYTE** で，内部での処理については集合型で扱うようなプログラムが作れます．**リスト1-4** がその例です．ここでは，まず **apple** から **lemon** までの値を持つ順序型 **fluit** を定義し，変数 **fluitbag** を **fluit** を元に持つ集合として定義しています．

さらに，**BYTE** 型変数 **fluitport** を **fluitbag** と同じアドレスに配置しています．集合型は **WRITELN** の対象にはなり得ませんが，**BYTE** 型なら実行結果のように出力できます．**リスト1-4** の例では，コンソールから確認できるように **WRITELN** を使っていますが，これを I/O ポートへの出力に置き換えれば，**図1-6** のような感じで各果物にくっついているランプを点灯させることができます．

● ビット出力のプログラム例

実用的なプログラムは**リスト1-5** のようにしたほうがよいでしょう．ここでは2種類の方法を例示しています．ひとつは手続き **outbit** で，果物のランプを消すかつけるかをパラメータ **onoff** で表しています．この中で **PORT** [n] は Turbo Pascal があらかじめ定義している入出力用配列で，代入されるときは出力に，値をみるときは入力になります．この例では **fluitport** が持っている **BYTE** を出力ポート **00E0H** へ出力しています．

別の方法としては，点灯用の **lighton** と消灯用の

〈リスト1-4〉集合型変数によるビット操作

```
PROGRAM bitio ; (* Bit In Out By SET OF *)
TYPE fluit = (apple,banana,cherry,peach,orange,mango,grape,lemon) ;
VAR  fluitbag : SET OF fluit ; fluitport : BYTE ABSOLUTE fluitbag ;
BEGIN
fluitbag:=[apple,cherry,orange,grape] ;
WRITELN('fluitport = ',fluitport) ;
fluitbag:=fluitbag-[grape,orange]+[peach,banana] ;
WRITELN('fluitport = ',fluitport) ;
END .
```

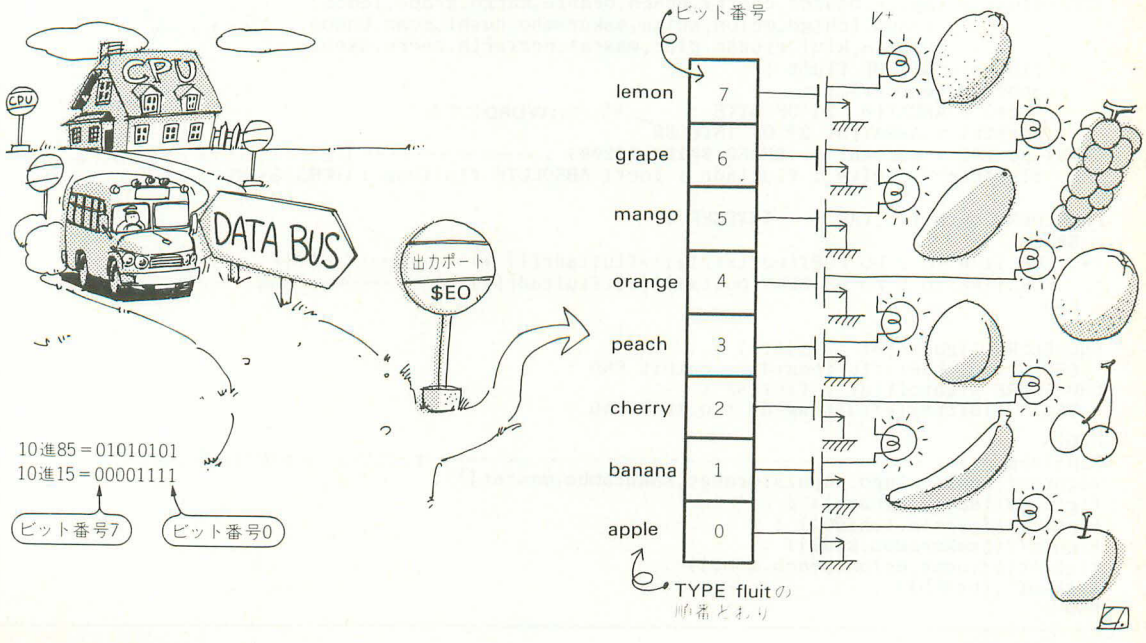
同じアドレスとなる

(a) プログラム

```
H>tt
fluitport = 85
fluitport = 15
```

(b) 実行結果

〈図1-6〉ビット出力の概念図



lightoff を用意するものがあります．どちらを使うかはフィーリングの問題です(両方混ぜて使うのは考えものですが)．**リスト1-5** のプログラムは，実行してもユーザのハードウェアが出力ポート **00E0H** につながっていなければ何の動作もしません．

ところで，この方式によるとビット数を増やしても16ビットの **WORD** 型の入出力(**PORTW**[**n**])までしか扱えません．さらに多くのビットを扱うには，集合変数をポートの数だけ用意しなければなりません．しかし，そうなると，**lighton** などのパラメータに，どの集合を操作するかを指定するものを追加しなければならず，見た目にもわずらわしくなります．また，ビット0はどのポートでも **apple** となってしまって，直感的に動作と結びつきにくくなります．やはり，各ビットに勝手な名前をつけて，そのビット名で8ビットのときと全く同じように扱えるのが理想です．

そこで奥の手ですが，まず集合の元となる順序型は256個の値まで許されているので，この範囲内で何個でもビットの名前を定義します．これを元とする集合の変数は**(n−1) DIV 8** バイトを占めるので，この数の **BYTE** 型の配列を集合変数と同じアドレスに配置します．これで集合変数の値を連続したポートに出力する

〈リスト1-5〉
実用のためのプログラム例

```
PROGRAM bitout ; (* Bit Out By SET OF *)
TYPE fluit = (apple,banana,cherry,peach,orange,mango,grape,lemon) ;
     fluits = SET OF fluit ;
     onoff = (on,off) ;
CONST allights : fluits =
              [apple,banana,cherry,peach,orange,mango,grape,lemon] ;
VAR  fluitbag : fluits ; fluitport : BYTE ABSOLUTE fluitbag ;

PROCEDURE outbit(dt : fluits ; f : onoff) ;
  BEGIN
  IF f=on THEN fluitbag:=fluitbag+dt ELSE fluitbag:=fluitbag-dt ;
  PORT[$E0]:=fluitport
  END ;

PROCEDURE lighton(dt : fluits) ;
  BEGIN fluitbag:=fluitbag+dt ; PORT[$E0]:=fluitport END ;
PROCEDURE lightoff(dt : fluits) ;
  BEGIN fluitbag:=fluitbag-dt ; PORT[$E0]:=fluitport END ;

BEGIN
fluitbag:=[] ;
outbit([apple,peach,grape],on) ; outbit([grape,orange],off) ;
lighton([banana,mango,lemon]) ; lightoff([peach]) ; lightoff(allights)
END .
```

〈リスト1-6〉連続しないポートへのビット出力操作

```
PROGRAM multibit ; (* Bit Out By SET OF for multi bytes *)
TYPE fluit = (apple,banana,cherry,peach,orange,mango,grape,lemon,
              ringo,ichigo,melon,suika,sakurambo,nashi,momo,budoo,
              papaia,kiui,ryugan,pine,mascat,necterin,zacro,akebi) ;  } 3バイト分の集合の名前を順序型て定義
     fluits = SET OF fluit ;
     onoff = (on,off) ;
     ioctl = ARRAY[0..2] OF BYTE ;        ← V5.0ではWORDにする
     porttbl = ARRAY[0..2] OF INTEGER ;
CONST portxt : porttbl = ($E0E0,$0100,$0200) ; ---------- I/Oポート(8ビット)のアドレス表
VAR  fluitbag : fluits ; fluitadr : ioctl ABSOLUTE fluitbag ; (連続しないポートに割り当てる)

PROCEDURE outbit ; VAR i : INTEGER ;
  BEGIN
(*  FOR i:=0 TO 2 DO PORT[portxt[i]]:=fluitadr[i] *) --------- (本番用)
    FOR i:=0 TO 2 DO WRITELN(portxt[i]:6,fluitadr[i]:6) ------- デバッグ用
  END ;

PROCEDURE lighton(dt : fluits) ;
  BEGIN fluitbag:=fluitbag+dt ; outbit END ;
PROCEDURE lightoff(dt : fluits) ;
  BEGIN fluitbag:=fluitbag-dt ; outbit END ;

BEGIN
fluitbag:=[] ; ---------- すべてのビットを消す(空集合を代入)
lighton([apple,ringo,papaia,orange,sakurambo,mascat]) ;
lightoff([apple,banana]) ;
lightoff([papaia,ichigo]) ;
lightoff([sakurambo,kiui]) ;
lighton([ryugan,melon,peach,momo]) ;
lightoff([peach]) ;
END .
```

ことができます(**FOR** 文でバイト数だけ繰り返す)．しかし，実際にはビット出力のポートが連続していないことも多いので，8ビットごとに全く異なるポートを指定できないものかと考えます．

そこでポート・アドレスの値を並べた配列の定数を作り，バイト出力時にアドレスの変換を行わせます．これがPascalでは簡単に書けてしまいます．この方式で24ビットの出力を行う例を**リスト1-6**に示します．このリスト中のビットの名前と**ARRAY**の大きさを変えれば，256ビットまでの不連続なポートへのビット操作が，ビット名のみを意識して書き表せるようになります．

Turbo Pascal V 3.0のキー定義

Turbo Pascalのパッケージには，**TINST**というユーティリティが付属しており，これによってCRTやキーボードの制御方式の変更ができるようになっています．特に，V 3.0では初期状態としてほとんど特殊キーを使用しておらず，機種依存度は極めて低くなっています．ちなみに，リコーのデータ・ベース専用マシン マイツール(MS-DOSを使っているが内部情報は公開されていない)でもエディタやコンパイラを動作させることができました．

とはいっても，例えばPC9801のうえで使うとすれば，矢印キーやDELキー，BSキーなどをその意味通りに使いたいのは当然で，そのために**TINST**が付属しているわけです．そこで，**TINST**を使ってキー定義を変更していくと，BSキーと⇦キーが同じコードを発生することがわかります．本来はBSキーを“Delete left character”として，⇦キーは“Character left“として使いたいのですが，この使い分けは不可能なのです．

この原因はTurbo Pascalにあるのではなく，MS-DOS(の中のIO.SYS)にあるようです．つまり，MS-DOSがユーザ・プログラム(この場合は**TINST**)に対してBSキーと⇦キーを区別できないようにしているのです．

ところで，MS-DOSにはキー入力から発生するコードを変更するためのユーティリティ**KEY.COM**が付属しており，これによって作られる**KEY.TBL**がシステム起動時に存在すると，自動的にキー定義が変更されるようになっています．そこで，これを使用してBSキーと⇦キーの区別ができるように設定しようと思いました…が．

何ともはや…(よくあることですが)キー定義変更ユーティリティ**KEY.COM**そのものがBSキーと⇦キーを区別する能力を持っていないのです(MS-DOSの上で動いているから…？)．それにしても困ったものなのですが困っていても仕方がないので最後の手段(本来は，こういう方法を使わなければならないのはシステムとしてよくないのですが)として，デバッガSymdebを使って**KEY.COM**によって作られた**KEY.TBL**の中の⇦キーに当たる部分を**1DH**に書き換えます．

この**KEY.TBL**のファイル内部のフォーマットについては公開されていませんが，デバッガで見てみるとだいたいの察しがつきます．このうちファンクション・キー以外の特殊キーについては図1-Aのようになっており，⇦キーは**025E**番地にあたります．

KEY.TBLは最初は存在していないので，とりあえず元のままに近い**KEY.TBL**を作る必要がありますが，同じ作るならということで，ファンクション・キーもPascal向きの**PROCEDURE**や**REPEAT UNTIL**などを定義して使っていますが，システム用の**AUTOEXEC.BAT**などを入れておくのもよいかも知れません．

KEY.TBLを変更した後，再起動してキー定義を有効にし，その上で**TINST**でTurbo Pascalのキー定義を変更すれば，BSキーと⇦キーを正しく動作させることができるようになります．もちろん，このための**KEY.TBL**は**TINST**を使うときだけでなく，Turbo Pascalを使うときには，必ず起動時に有効になっていなければなりません．キー定義の変更を確認するためにも，ファンクション・キーの一部は変更しておいたほうがよいようです．

〈図1-A〉
KEY.TBLの変更

```
symdeb a:key.tbl
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is [80286]
-d240 27f
3ED4:0240  15 00 00 00 00 00 10 00-00 00 00 00 1B 50 00 00  .............P..
3ED4:0250  00 00 1B 44 00 00 00 00-0B 00 00 00 00 00 1D 00  ...D............
3ED4:0260  00 00 00 00 0C 00 00 00-00 00 0A 00 00 00 00 00  ................
3ED4:0270  18 00 00 00 00 00 1C 00-00 00 00 00 02 00 00 00  ................
-
```

第1回 技術者のためのdBASE III プログラミング入門

メニュ・プログラムの作成

ASSIST HELP MENU QUIT

丹羽 一夫

パソコンも，今や，手軽に使える身近な道具としてなくてはならないものになってきました．それもこれも，ワープロ/表計算/データ・ベースといった実務に役立つ分野のソフトウェアが豊富に提供されており，けっこう仕事に使えるからでしょう．

この連載では，その中のデータ・ベース・ソフトを使って仕事をしてみることにします．私たちのまわりにはいろいろなデータがあふれていますが，大げさなものではなく，仕事の現場にころがっている身近なデータを処理してみようというのが，その目的です．

このようなデータは，たとえばリポートを書くのにワープロ・ソフトで処理したのではたんなる数字や文字列にしか過ぎませんが，データ・ベース・ソフトを使って処理するとそれは数値データや文字データとなり，より役立つ情報を私たちに提供してくれます．

この連載では実例をあげて話をすすめていきますが，そこからくみとっていただきたいのは，仕事の手順や考え方についてです．とりあえずお見せする実例を試してみたら，データをあなたのお持ちのものに置き換え，またプログラムをあなたの場合に書き換えて自分のものにしていただければと思っています．

なお，この連載をすすめていくにあたり，パソコンについてのひととおりの知識や経験をお持ちで，データ・ベース・ソフトのdBASE IIIを起動したことがある，ということを前提にさせていただきます．

準備と目標

データ・ベース・ソフトとしては，タイトルでもおわかりのように，日本アシュトン・テイト社のdBASE IIIを使います．

NECが出しているPC98シリーズの“アプリケーション情報”を見ると，なんと100種類以上のデータ・ベース・ソフトが紹介されています．その中で，このdBASE IIIを選んだのは，歴史が古くて完成度が高いということもありますが，簡易言語といっていいほど多くのコマンドや関数が用意されていて多彩なプログラムが組めるからです．

● dBASE IIIにもいろいろあるが

ところで，とりあえずdBASE IIIといいましたが，実際にはそのほかにdBASE III PLUS，それにサザンパシフィック社から発売されているdBASE IIIクローンのDBXLといったものがあり，dBASE III用に作ったプログラムは基本的にはどれでも走ります．そこで，ソフトウェアとしてはdBASE III，dBASE III PLUS，DBXLの三つを対象とすることにします．

● プログラムを作るためのテキスト・エディタも用意したい

ソフトウェアの話をしたついでにもう一つ，プログラムを組むときのために，何かエディタを用意してください．dBASE IIIには**MODIFY COMMAND**というテキスト・エディタも内蔵されていますが，行数に制限があるなど まあ補助用といったところです．

なお，エディタはMIFESでもREDでも，あるいはワープロ(テキスト・ファイルに落とすこと)でもかまいません．

ハードウェアはPC9801シリーズを使いますが，dBASE IIIやdBASE III PLUS，DBXLはほかの機種用のものも用意されていますから，たぶん同じようにやれるはずです．もしトラブルが発生しても，少しの手直しでOKとなるでしょう．

● dBASEのプログラム言語を使って開発する

ソフトウェアとハードウェアの準備がすんだところで，このあとの目標をたてることにしましょう．

まず，dBASE IIIやDBXLで仕事をする場合のやり

〈連載の予定〉

(1) 準備をしたり目標を定めて，メニュ・プログラムを作る．
(2) データ入力用のプログラムを作り，データを入力する．
(3) データ訂正用のプログラムを作り，訂正や削除ができるようにする．
(4) データを加工していろいろな情報を提供する，データを利用するためのプログラムを作る．
(5) データの一覧表を印刷する．
(6) RS-232Cを通してデータを収集する方法を試みる．

方には，直接コマンドを打ったり **ASSIST** や **INTRO** を利用する方法と，プログラムを組む方法の二つがあります．

このうち，前者の方法で仕事ができるのは，自分の持っているデータを自分自身で処理する場合で，それもある程度定形的な処理しかできません．

この連載では，後者のプログラムを組む方法で仕事をしてみることにします．この方法だと，dBASE IIIを起動したときに表れるオープニング画面を除けば，dBASE IIIをまったく意識することなく仕事を開始して終了することができますし，ほかの人でも同じように仕事のできる環境が実現できます．

なお，プログラムを組むのは一見面倒なようですが，Basic のプログラムを組んだ経験のある方ならすぐになれるはずです．

● まず身近なデータとして「気温」を用いる

つぎに，どんなデータを用意して，それをどのようにするかを決めておきましょう．

実例とするデータは連載の進行につれていくつか用意する予定ですが，最初は身近なわかりやすいものということで，気温のデータを使ってみることにしました．くわしいことはあとで説明しますが，「東京・4月の気温」が最初のサンプル・データです．

用意するデータがこのように決まったら，このデータに対してどのような仕事があるかをつぎに考えます．「東京・4月の気温」というデータに対しては，データ・ベースの作成(データの入力/訂正)，データ・ベースの利用(各種情報の提供，一覧表の印刷など)，といったことを目標にしてみることにします．

最初に「仕事名」と「略称」を決める

ハードウェアとソフトウェアの準備ができ，処理しようとするデータが決まったら，いよいよ仕事にとりかかることになります．

ここでまずやっておかなければならないのは，「仕事

〈表1-1〉(1)
東京・4月の気温一覧表

01日	03時	11.9度	07日	03時	11.2度	13日	03時	11.1度	19日	03時	16.0度	25日	03時	12.2度
	06時	11.0度		06時	11.2度		06時	11.0度		06時	15.8度		06時	13.3度
	09時	14.6度		09時	15.4度		09時	16.5度		09時	19.0度		09時	15.3度
	12時	18.7度		12時	18.0度		12時	21.0度		12時	21.0度		12時	17.0度
	15時	19.6度		15時	18.3度		15時	22.9度		15時	21.9度		15時	17.2度
	18時	20.0度		18時	16.7度		18時	19.6度		18時	20.2度		18時	15.4度
	21時	17.9度		21時	15.3度		21時	17.6度		21時	18.2度		21時	13.3度
	24時	12.5度		24時	15.3度		24時	15.6度		24時	17.3度		24時	11.8度
02日	03時	10.5度	08日	03時	13.6度	14日	03時	12.2度	20日	03時	15.9度	26日	03時	11.0度
	06時	8.9度		06時	11.5度		06時	11.4度		06時	14.1度		06時	11.8度
	09時	12.6度		09時	12.5度		09時	18.7度		09時	19.0度		09時	14.8度
	12時	14.4度		12時	14.7度		12時	22.3度		12時	22.1度		12時	17.0度
	15時	16.6度		15時	14.1度		15時	21.8度		15時	21.7度		15時	18.4度
	18時	15.5度		18時	15.9度		18時	20.4度		18時	20.0度		18時	18.0度
	21時	14.3度		21時	12.9度		21時	17.9度		21時	19.7度		21時	14.0度
	24時	13.9度		24時	13.3度		24時	17.6度		24時	19.5度		24時	13.7度
03日	03時	11.4度	09日	03時	13.7度	15日	03時	17.3度	21日	03時	14.8度	27日	03時	12.0度
	06時	12.5度		06時	13.7度		06時	16.9度		06時	14.2度		06時	12.1度
	09時	12.9度		09時	18.6度		09時	18.8度		09時	16.4度		09時	17.3度
	12時	14.6度		12時	21.6度		12時	19.8度		12時	18.6度		12時	22.0度
	15時	13.6度		15時	22.6度		15時	18.6度		15時	17.3度		15時	21.0度
	18時	14.3度		18時	18.8度		18時	16.3度		18時	15.4度		18時	19.5度
	21時	14.2度		21時	17.3度		21時	16.3度		21時	15.0度		21時	17.2度
	24時	12.7度		24時	16.4度		24時	16.5度		24時	14.4度		24時	14.7度
04日	03時	12.2度	10日	03時	15.9度	16日	03時	16.1度	22日	03時	14.2度	28日	03時	14.1度
	06時	11.5度		06時	15.6度		06時	16.5度		06時	13.1度		06時	13.4度
	09時	13.7度		09時	16.6度		09時	17.6度		09時	17.1度		09時	14.3度
	12時	17.0度		12時	17.4度		12時	21.1度		12時	19.3度		12時	14.7度
	15時	18.2度		15時	17.2度		15時	23.2度		15時	19.0度		15時	13.2度
	18時	15.8度		18時	16.0度		18時	20.1度		18時	17.9度		18時	11.3度
	21時	14.4度		21時	15.1度		21時	19.1度		21時	17.3度		21時	9.9度
	24時	12.8度		24時	13.5度		24時	17.2度		24時	17.2度		24時	9.2度
05日	03時	11.6度	11日	03時	13.7度	17日	03時	16.9度	23日	03時	16.8度	29日	03時	8.7度
	06時	9.7度		06時	13.3度		06時	14.8度		06時	16.7度		06時	8.9度
	09時	9.6度		09時	14.5度		09時	19.0度		09時	16.9度		09時	13.5度
	12時	9.8度		12時	17.8度		12時	19.7度		12時	17.1度		12時	16.9度
	15時	11.1度		15時	17.6度		15時	21.2度		15時	16.1度		15時	17.3度
	18時	10.2度		18時	14.5度		18時	20.2度		18時	15.8度		18時	15.2度
	21時	10.4度		21時	11.3度		21時	18.0度		21時	14.7度		21時	14.1度
	24時	11.1度		24時	10.4度		24時	15.5度		24時	14.9度		24時	12.6度
06日	03時	9.9度	12日	03時	8.2度	18日	03時	14.2度	24日	03時	14.1度	30日	03時	10.9度
	06時	8.7度		06時	7.6度		06時	13.2度		06時	14.6度		06時	11.7度
	09時	13.3度		09時	11.1度		09時	18.4度		09時	17.3度		09時	16.5度
	12時	16.6度		12時	14.2度		12時	20.7度		12時	19.7度		12時	20.1度
	15時	18.3度		15時	15.8度		15時	22.1度		15時	18.7度		15時	19.6度
	18時	14.4度		18時	13.5度		18時	20.1度		18時	15.7度		18時	18.3度
	21時	13.6度		21時	11.0度		21時	17.8度		21時	13.5度		21時	16.7度
	24時	12.9度		24時	10.3度		24時	16.8度		24時	13.3度		24時	16.2度

名」と「略称」を決めることです.

まず仕事名ですが，すでにお話したように最初の仕事は東京の4月の気温を取り上げることにしましたから，「東京・4月の気温」とすることにしましょう.

この仕事名は，このあとメニュやその他のプログラムを組むときに必要になり，仕事のつどCRTの画面に表示されることになります.

また，一覧表を作成するときにも，タイトルとして最初に印刷します.

つぎは，略称です．略称は各種のファイルを作るときのファイル名に使いますが，ここではとりあえず，「**TEMP**」と「**TP**」の二つを用意することにします．もちろん，**TEMP**はTemperatureを略したもので，**TP**は**TEMP**をさらに略したものです．なお，**TEMP**以外に**TP**も用意したのは，ファイル名として利用するときに**TEMP**でもまだ長過ぎることがあるからです.

データの準備

dBASE IIIのデータは最終的にはフロッピ・ディスクやハード・ディスクの上に作ったデータ・ベース・ファイル(以下，**DBF**ファイル)に納めることになります．そこまでの手順としては，とりあえず二つの方法が考えられます.

その一つは，紙の上に用意されたデータをキーボードから入力する方法，もう一つはRS-232Cなどを通して直接データをパソコン内に取り入れる方法です.

● 元データの用意

この連載では後者の方法もやってみる予定ですが，「東京・4月の気温」のデータは前者の方法で用意してみることにしました.

さて，ここで用いる気温のデータは，たんなるサンプルですから適当に作ってしまってもいいのですが，それでは面白くないので，実際のデータを手に入れてみることにしました．そこで行ってみたのが，東京・大手町にある気象庁天気相談所です．ここでこちらの希望を説明し，気温の変化がいちばん大きい月をきいてみたらそれは4月とのこと，そこで1989年4月1日～30日の東京の“地上気象観測日原簿”というのを手に入れました.

ちなみにこの日原簿には，3時間おきの気圧，気温，湿度，風向/風速，日照時間，降水量，積雪の深さ，雲量といったものが示されています.

表1-1は，この日原簿の中の気温だけを書き出してみたものです．これが，これから実例として使用する「東京・4月の気温」です.

● データ・ファイルの作成

では，このデータをdBASE IIIに取り込むためのDBFファイルを作りましょう．DBFファイルの作成に使用するコマンドは**CREATE**，**表1-2**にこのファイルのデータ・ベース構造を示します．ファイル名は**TEMP8904.DBF**で，これは1989年4月のデータだということを表します.

新しく**DBF**ファイルを作るときには，フィールド名，フィールド・タイプ(型式)，それにフィールド幅と，数値型のときには小数の数を決めなければなりません.

〈表1-2〉一番基本となるDBFファイルの構造

```
データベースの構造    : A:TEMP8904.dbf
データ・レコードの数 :          0
最終更新日付          : 09/05/89
番号    フィールド  タイプ      幅    小数
    1   DD          文字型      2
    2   TT          文字型      2
    3   TEMP        数値型      5
< 合計 >                        10
```

〈表1-3〉月のフィールドを追加する

```
データベースの構造    : A:TEMP89.dbf
データ・レコードの数 :          0
最終更新日付          : 09/05/89
番号    フィールド  タイプ      幅    小数
    1   MM          文字型      2
    2   DD          文字型      2
    3   TT          文字型      2
    4   TEMP        数値型      5
< 合計 >                        12
```

〈表1-4〉年のフィールドを追加する

```
データベースの構造    : A:TEMP.dbf
データ・レコードの数 :          0
最終更新日付          : 09/05/89
番号    フィールド  タイプ      幅    小数
    1   YY          文字型      2
    2   MM          文字型      2
    3   DD          文字型      2
    4   TT          文字型      2
    5   TEMP        数値型      5
< 合計 >                        14
```

● フィールド(項目)を決める

表1-2では，まずフィールド名は日を**DD**，時間を**TT**，そして気温を**TEMP**としてあります.

表1-1で用意したデータは1989年4月の1ヵ月間のものでしたから**表1-2**のようなものでよかったのですが，もし1年間のデータを扱いたかったら，**表1-3**のように月のデータ(**MM**)を加えてやります．ファイル名の**TEMP89**は，1989年を表しています．そして，**表1-4**のように年のデータ(**YY**)を加えると，何年にもわたってのデータの管理が可能になります.

● フィールド・タイプを決める

つぎはフィールド・タイプですが，「東京・4月の気温」のように日時も含めてデータがすべて数字の場合には数値型にするか文字型にするか，ちょっと迷いま

す．まあ，大まかな感じとしては，計算をするようなフィールドは数値型でなければなりませんし，インデックス・ファイル(以下，**NDX** ファイル)のキーに使ったり印刷のスタイルを作ろうとするようなときには文字型のほうが扱いやすい，といったようなこともあります．

今回**表1-2** のように決めたのは，日時は数字を使いますがその扱いは数値というよりも文字であるということ，一方，気温のほうは数値そのもので，あとで数値としていろいろ利用してみるつもりだからです．

なお，文字型とした日時についても必要があれば **VAL** 関数を使って数値に直して使うこともできますし，気温についても **STR** 関数を使って文字列に直すこともできますから，そんなに心配はいりません．

● **フィールドの幅はあとでも変更ができる**

フィールド幅については，今回のようにデータの文字数がきちんと決まっている場合はいいのですが，そうでない場合にはどう決めるか ちょっとやっかいです．文字数が決まらない場合には 最大値を予測して用意することになりますが，もし途中で不足したり最終的に大きく余りが出た場合には，**MODIFY STRUCTURE** コマンドを使ってデータ・ベースの構造を手直しすることも可能です．

これで，データの入れ物ができました．**CREATE** コマンドからいきなり入力を始めることもできますし，いったん終了したあとであれば，**APPEND**(データの追加)コマンドを使ってダイレクトにデータを入力することもできます．でも，楽しみはあとにとっておくことにして，いったん **QUIT**(終了)ということにしましょう．

メニュ・プログラムの作成

レストランに入ってテーブルにつくと，まずメニュがでてきます．まあ，料理の種類が少ないとか，何を食べるかがすでに決まっている場合にはメニュのいらないときもありますが，いろいろ選んで食べたいというときにはメニュは便利です．

というわけで，「東京・４月の気温」についても，メニュを用意してみることにします．このメニュは，データを複数の人で管理したり利用する場合には特に有効です．

〈リスト1-1〉一番基本的なメニュ・プログラム

```
*「東京・４月の気温」ﾒﾆｭｰ    <TPMENU1>    1989/9    by Kazuo Niwa ------------
SET TALK OFF──────────────────────────────①
*--------------
DO WHILE .T.      ← 永久ループ
  CLEAR           ← 画面クリア
  *
  @  2,26 SAY "＊ 「東京・４月の気温」 ＊"
  @  3,26 SAY "--------------------------"
  @  5,35 SAY "メ ニ ュ ー"
  @  8,32 SAY "１…データの入力"
  @ 10,32 SAY "２…データの訂正"
  @ 12,32 SAY "３…データの利用"
  @ 14,32 SAY "４…一覧表の印刷"
  @ 16,32 SAY "５…終      了"
  @ 19,1
  INPUT SPACE(26)+"仕事の番号を選んでください。" TO NO
  *------------
  DO CASE ──────────────┐
    CASE NO=1                 │
      @ 22,34 SAY "データの入力"  │
      WAIT                    │
    CASE NO=2                 │
      @ 22,34 SAY "データの訂正"  │
      WAIT                    │
    CASE NO=3                 ├──②
      @ 22,34 SAY "データの利用"  │
      WAIT                    │
    CASE NO=4                 │
      @ 22,34 SAY "一覧表の印刷"  │
      WAIT                    │
    CASE NO=5                 │
      QUIT ← dBASEを終了する     │
  ENDCASE ──────────────┘
ENDDO
*------------------------------------ END ------------------------------------
```

◆ 仕事の内容を確定する

メニュの役目は，自分のやりたいと思う仕事をメニュから選択し，その仕事に動作を移すことです．そのようなわけで，メニュを作るには，まずどんな仕事をするかを確定させなければなりません．

では，「東京・４月の気温」のためのメニュを作ってみましょう．仕事の内容から，メニュは，

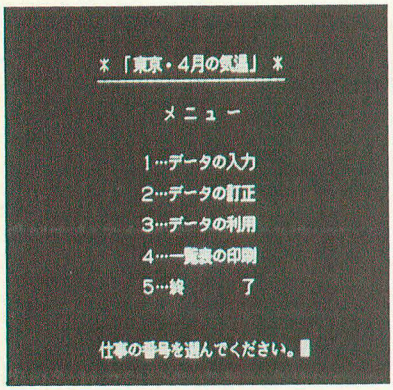
① データの入力 **<TPINP.PRG>**
② データの訂正 **<TPCOR.PRG>**
③ データの利用 **<TPUSE.PRG>**
④ 一覧表の印刷 **<TPPRI.PRG>**
⑤ 終　　　了

ということにしてみます．

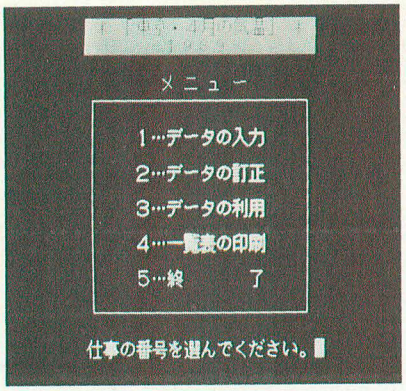
仕事の内容を決めたら，ついでにそれぞれのプログラム名も決めておきましょう．右側に示したのがそれで，プログラムの頭には略称として用意した **TP** をつけて「東京・４月の気温」用のプログラムであることを示します．

◆ メニュ・プログラムの実際

dBASE IIIのプログラムに限ら



〈写真1-1〉TPMENU1の画面



〈写真1-2〉TPMENU2プラス枠の画面

〈リスト1-2〉実際のプログラムへ分岐する部分

```
DO CASE
  CASE NO=1
    DO TPINP
  CASE NO=2
    DO TPCOR
  CASE NO=3
    DO TPUSE
  CASE NO=4
    DO TPPRI
  CASE NO=5
    CLEAR
    CANCEL
ENDCASE
```

ず，BASICでも，同じ仕事に対して100人の人がプログラムを組んだら100通りのプログラムができるといわれます．そのようなわけで，このあとお見せするプログラムは，すべてその一例であるということを頭に入れておいてください．

では，「東京・4月の気温」のメニュ・プログラム〈**TPMENU.PRG**〉を作ってみることにしましょう．

リスト1-1は，CRTディスプレイの表示画面に装飾をまったくほどこしてない，骨組みだけのようなプログラムです．このあとメニュ・プログラムは何本か作るので，その最初ということで，プログラム名は〈**TPMENU1.PRG**〉としてあります．

まず，**リスト1-1**の①は手紙の書き出しでいえば“拝啓”といった決まり文句で，メニュ・プログラムの場合には，仕事全体にかかわる各種の設定や割り込みの条件などをここに書いておきます．

リスト1-1をみると，メニュ・プログラムの本体は**DO WHILE～ENDDO**のループの中に入れてあります．このようにするのは，メニュと仕事の間を行ったり来たりできるようにするためです．このようにしておけば，仕事を終わったあとはかならずメニュに戻ってくることができます．なお，仕事を完全に終わるときには，「5」の終了を選びます．

それでは**リスト1-1**の〈**TPMENU1.PRG**〉をエディタで作り，dBASE IIIを起動したあとプログラムを走らせてみてください(**DO TPMENU1**)．

仕事を選択すると，その結果が**写真1-1**のように表示されたでしょう．まだ仕事をするためのプログラムが用意されていないので，ちょっとメニュ・プログラムで遊んでみたというわけです．**リスト1-1**の②の部分がこの遊びであることはおわかりでしょう．

リスト1-1の②の部分は，最終的には**リスト1-2**のように書き直します．

これで，最初のメニュ・プログラム〈**TPMENU1.PRG**〉ができました．このプログラムは白1色で，最初にお話したように装飾もありません．でも，仕事だけは1

〈リスト1-3〉表示色に工夫をこらしたメニュ・プログラム

```
*「東京・4月の気温」ﾒﾆｭｰ    <TPMENU2> 1989/9 by Kazuo Niwa ---
SET TALK OFF
*----------------
DO WHILE .T.
  CLEAR
  *----------------
  SET COLOR TO GBI
  @  2,26 SAY " ＊ 「東京・4月の気温」 ＊ "
  SET COLOR TO GRI
  @  3,26 SAY "          - 1 9 8 9 -          "
  SET COLOR TO R
  @  5,35 SAY "メ ニ ュ ー"
  SET COLOR TO W
  *----------------
  I=7
  SET COLOR TO G
  @  6,26 SAY "┌──────────────┐"
  DO WHILE I<18
    @ I,26 SAY "|                |"
    I=I+1
  ENDDO
  @ 18,26 SAY "└──────────────┘"
  *----------------
  SET COLOR TO W
  @  8,32 SAY "1…データの入力"
  @ 10,32 SAY "2…データの訂正"
  @ 12,32 SAY "3…データの利用"
  @ 14,32 SAY "4…一覧表の印刷"
  SET COLOR TO GB
  @ 16,32 SAY "5…終　　了"
  @ 19,1
  SET COLOR TO GR
  INPUT SPACE(26)+"仕事の番号を選んでください。" TO NO
  SET COLOR TO W*
  *----------------
  DO CASE
    CASE NO=1
      @ 22,34 SAY "データの入力"
      WAIT
    CASE NO=2
      @ 22,34 SAY "データの訂正"
      WAIT
    CASE NO=3
      @ 22,34 SAY "データの利用"
      WAIT
    CASE NO=4
      @ 22,34 SAY "一覧表の印刷"
      WAIT
    CASE NO=5
      QUIT
  ENDCASE
ENDDO
*---------------------------------- END ----------------------
```

N:黒　R:赤
B:青　BR:マゼンタ
G:緑　GR:黄
BG/GB:シアン　W:白
X:ブランク

I:反転させる
＊:ブリンクさせる

①

②

人前にやりますから，モノクロやラップトップ機で使うといいでしょう．

さて，最初のメニュ・プログラムは地味にまとめましたので，今度は思いきって派手なプログラムを組んでみることにします．

リスト1-3 は，そのプログラム〈**TPMENU2.PRG**〉です．このプログラムを走らせると，**写真1-2** のようになります．色がついているところをお見せできないのが残念ですが…．そういえば，あちこちに **SET COLOR TO** という色指定のコマンドが目につくでしょう．

リスト1-3 の① の部分は，画面の中の枠を作っているところです．dBASE IIIの場合にはキャラクタとして用意されているケイ線で枠を作らなければなりませんが，dBASE III PLUS や DBXL の場合には，この部分を**リスト1-4** のように書き直すことによって枠が作れます．なお，DBXL の場合の色の指定は，たんに **SET COLOR TO G** とすればOK です．

これでもう一つメニュ・プログラムができましたが，〈**TPMENU2.PRG**〉のほうも，このあと実際の仕事に使うときには② のところを**リスト1-2** のように書き直します．

〈リスト1-5〉 ヘルプを出すようにした

```
*「東京・４月の気温」ﾒﾆｭｰ    <TPMENU3>    1989/9     by Kazuo Niwa ----
*ヘルプつき
SET TALK OFF
*--------------
DO WHILE .T.←――Ⓐ
  CLEAR
  *
  @  2,26 SAY "＊ 「東京・４月の気温」 ＊"
  @  3,26 SAY "----------------------------"
  @  5,35 SAY "メ ニ ュ ー"
  @  8,32 SAY "１…データの入力"
  @ 10,32 SAY "２…データの訂正"
  @ 12,32 SAY "３…データの利用"
  @ 14,32 SAY "４…一覧表の印刷"
  @ 16,32 SAY "５…終　　了"
  @ 20,26 SAY "仕事の番号を選んでください。"――――――――――①
  *--------------
  @ 22,26 SAY "ヘルプ→「ＨＥＬＰ」を押す。"
  DO WHILE .T.
    H=INKEY()
    IF H=28
      CLEAR
      @ 10,32 SAY "「ヘルプ」です。"
      @ 19,1
      WAIT SPACE(21)+"何かキーを押すと，メニューに戻ります。"
      EXIT
    ENDIF                          ――②
    *
    IF H>=49 .AND. H<=53
      NO=VAL(CHR(H))
      EXIT
    ENDIF
  ENDDO
  *
  IF H=28
    LOOP←――Ⓐの DO WHILE 文にもどる
  ENDIF
  @ 22,1
  *------------
  DO CASE
    CASE NO=1
      @ 22,34 SAY "データの入力"
      WAIT
    CASE NO=2
      @ 22,34 SAY "データの訂正"
      WAIT
    CASE NO=3
      @ 22,34 SAY "データの利用"
      WAIT
    CASE NO=4
      @ 22,34 SAY "一覧表の印刷"
      WAIT
    CASE NO=5
      QUIT
  ENDCASE
ENDDO
*---------------------------------- END ----------------------------
```

Hの値	数値
49	1
50	2
51	3
52	4
53	5

◆「ヘルプ」とデータのバックアップ

今まで紹介してきたメニュ・プログラムでメニュとしての仕事は十分にこなしてくれるのですが，メニュのところでやれるようにしておくと便利なものがいくつかあります．その一例が，必要に応じて仕事についてのメッセージを出す「ヘルプ」と，データのバックアップです．

では，**リスト1-1** のプログラムに手を加えて，「ヘルプ」を出してみることにしましょう．

リスト1-5 の〈**TPMENU3.PRG**〉がそれで，まず① の部分が**リスト1-5** と変わっています．

② が「ヘルプ」を出すところで，**DO WHILE** のループの中ではどのキーが押されるかを INKEY 関数が監視しています．そして，HELP キーが押されると「ヘルプ」を表示し，1～5 のキーが押されるとメニュの仕事に移ります．

なお，**リスト1-1** では仕事の番号を INPUT コマンドで取り入れていたのでリターン・キーを押して仕事を進めましたが，**リスト1-5** のほうは INKEY 関数を使っているため，1～5 のキーが押されると，いきなりつぎの仕事に移るところが変わっています．

それから，dBASE IIIではファンクション・キーの F1 に HELP が割り当てられており，この F1 を押しても同じように働きます．

うまくいったら，**リスト1-3** にも「ヘルプ」機能を組み込んでみてください．このほうは，ヘルプ画面もきれいに飾ってやるとよいでしょう．

つぎに，同じく**リスト1-1** に手を加えて，データのバックアップができるようにしてみましょう．

〈リスト1-4〉 画面に枠を出すコマンド

```
SET COLOR TO ,,G
@  6,26 TO 18,54 DOUBLE
```

リスト1-6の〈TPMENU4.PRG〉がそれで，①と②のところが追加になっています．

まず，②がデータをバックアップするところで，いったんバックアップするかどうかを聞いてくるようになっています．ここでもしNを入力すれば，バックアップせずにそのまま仕事を終わります．

もしYを入力すると，バックアップ作業に移ります．この例ではAドライブの〈TEMP8904.DBF〉をBドライブにコピーしますが，もしBドライブにディスケットが入っていないとエラーになりプログラムは止まってしまいますから注意が必要です．

なお，このバックアップでは，2回目以降は前のDBFファイルに上書きする形になります．そこで，①が必要になります．この①がないと，そのつど上書きしていいかどうかを聞かれます．

〈リスト1-6〉データのバックアップをとれるようにした

```
*「東京・４月の気温」ﾒﾆｭｰ    〈TPMENU4〉    1989/9      by Kazuo Niwa ------------
*データのバックアップつき
SET TALK OFF
SET SAFETY OFF ──────────────────────────────────①
*-------------
DO WHILE .T.
  CLEAR
  *
  @  2,26 SAY "＊ 「東京・４月の気温」 ＊"
  @  3,26 SAY "--------------------------"
  @  5,35 SAY "メ ニ ュ ー"
  @  8,32 SAY "１…データの入力"
  @ 10,32 SAY "２…データの訂正"
  @ 12,32 SAY "３…データの利用"
  @ 14,32 SAY "４…一覧表の印刷"
  @ 16,32 SAY "５…終　　了"
  @ 19,1
  INPUT SPACE(26)+"仕事の番号を選んでください。" TO NO
  *------------
  DO CASE
    CASE NO=1
      @ 22,34 SAY "データの入力"
      WAIT
    CASE NO=2
      @ 22,34 SAY "データの訂正"
      WAIT
    CASE NO=3
      @ 22,34 SAY "データの利用"
      WAIT
    CASE NO=4
      @ 22,34 SAY "一覧表の印刷"
      WAIT
    CASE NO=5
      @  5,1 CLEAR ──────────────────────────┐
      @ 10,1                                 │
      WAIT SPACE(20)+"データをバックアップしますか（Y OR N）" TO YN  ──②
      IF UPPER(YN)="Y" .OR. YN="ｿ"           │
        @ 11,1                               │
        @ 10,1                               │
        WAIT SPACE(11)+"ドライブＢにディスケットを入れてください。ＯＫならリターン"
        USE TEMP8904                         │
        COPY TO B:TEMP8904                   │
      ENDIF ─────────────────────────────────┘
      *
      CLEAR
      QUIT
  ENDCASE
ENDDO
*---------------------------------- END ----------------------------------
```

◆ 参考・引用*文献 ◆

(1)*地上気象観測日原簿，気象庁発行．

(2) dBASE III PLUS ユーザーズ・マニュアル リファレンス編，日本アシュトン・テイト㈱．

Keyword

データ・ベース　そのままでは利用できない生データを一定の規則で加工し，利用者が必要な情報を得やすいように，検索プログラムなどを提供した一連のシステムのこと．データの保存の方法には様々な方法が考案されているが，最近のパソコンではリレーショナル型と呼ばれるものが主流となっている．

dBASE III　OSがCP/Mの時代に産声をあげた歴史の長いリレーショナル・データ・ベース言語．米国ではVer IVが出ているが，国内ではIII plusがもっとも新しい．フィールドは固定長．plusになってインデックス・ファイルの作成が高速化され，ハード・ディスクやRAMディスクの併用によって，実用になるデータ量も飛躍的に増大した．発売元は日本アシュトン・テイト㈱．定価は268,000円(機種別にリリースされている)．

郵便はがき

料金受取人払

豊島局承認

2432

差出有効期間
1990年9月30日まで

切手をはらずにお出しください

170

(受取人)

東京都豊島区巣鴨1-14-2

CQ出版株式会社

トラ技コンピュータ リーダーズカード 行

読者アンケート

■ トラ技コンピュータ11月号に対する感想を率直にご記入ください.

資料請求カード　TC　1989年11月号

当社記入欄

1 VOL M001

2 受付

3 登録内容変更箇所No.

カード有効期限：1989年12月末日迄
送付先は勤務先に限ります。

4 請求状況　1＝新規 2＝登録済 3＝登録内容の変更

5 会員番号 T

トランジスタ技術の会員番号をご利用下さい。

6 フリガナ

7 姓名　姓　名

8 生年月日　19　年　月　日

9 〒　都道府県　区市郡　区町村

10 勤務先住所

11 会社名

12 部署名

13 勤務先電話番号　市外局番　局番　電話番号　(　)

14 役職

15 所属部門

16 担当分野

17 業種

18 従業員数

19 広告資料請求欄　請求No.の欄に資料請求番号を記入し、A. B. C. のご希望の項目に○をつけてください

請求No. / 希望項目										
購入希望	A	A	A	A	A	A	A	A	A	A
説明希望	B	B	B	B	B	B	B	B	B	B
資料希望	C	C	C	C	C	C	C	C	C	C

請求No. / 希望項目										
購入希望	A	A	A	A	A	A	A	A	A	A
説明希望	B	B	B	B	B	B	B	B	B	B
資料希望	C	C	C	C	C	C	C	C	C	C

資料請求カード記入法

専門誌の広告は重要な技術情報です。その中で読者の皆様が入手したい情報は、資料請求カードのご利用で、居ながらにしてお手元に届きます。ぜひ有効にご活用下さい。

- 必ず全項目にご記入下さい。
- 姓名・勤務先・電話番号・住所は所定の様式通り1枠ごとに書き入れて下さい。
- 会社名は株式会社ならば(株)、有限会社は(有)、財団法人は(財)と略し社名を書いて下さい。
- 生年月日は西暦で書いて下さい。
- 当カードは登録することができます。まず全項目にご記入の上投函下さい。当社より登録完了はがきが到着します。
- 当社請求カードの会員番号がある方は、会員番号・姓名・生年月日の記入ですみます。また会員番号をお持ちで登録内容に変更がある方は変更箇所を上記の他に書いて下さい。
- 役職・職種・専門分野・業種・従業員数は下記の該当する番号を記入して下さい。

役職
1. 一般職
2. 主任職
3. 係長職
4. 課長職
5. 部長　次長職
6. 役員職
7. 経営者
8. 教職(小、中、高校)
9. 教職(大学、専門学校)
10. 専門職(弁護士・医師・会計士等)
99. その他

所属部門
1. 研究・開発部門
2. 設計・技術部門
3. 生産・生産管理部門
4. 保守サービス部門
5. その他の技術部門
6. 営業部門
7. 購買・資材部門
8. 総務・経理部門
9. 人事・教育部門
10. 経営部門
99. その他

担当分野
1. コンピュータ(ハードウェア)技術
2. コンピュータ(ソフトウェア)技術
3. 半導体・デバイス技術
4. 電子回路技術
5. 計測・制御技術
6. 通信関連技術
7. 物性・材料
8. 機構設計
9. システムエンジニアリング
10. 検査・品質管理
11. 医師
12. 経営管理
13. 商工サービス
14. 教職
15. 学生
99. その他

業種
1. 建設・不動産
2. 金融・保険
3. 化学・エネルギー
4. 鉄鋼・金属
5. 機械・精密機器
6. 電子機器・部品
7. 輸送用機器・運輸
8. その他の製造
9. 商社・卸売
10. 通信・情報
11. 情報処理
12. その他のサービス
13. 政府公共機関
14. 学校・研究所・病院
15. 農林・水産業
99. その他

従業員数
1. 10人以下
2. 11人～50人
3. 51人～100人
4. 101人～500人
5. 501人～1000人
6. 1001人～5000人
7. 5001人～10,000人
8. 10,001人以上

IBIX

技術製品宣言。

大容量化時代に対応した
デバイスドライバ登場。

■PC-9801用拡張ディスクシステム■

メガ300

実績

メガ300はPC9801のMSDOSからそのまま300メガバイトの大容量固定ディスクをアクセスできるようにしたサブシステム。現在、パソコンの応用分野が、32ビット化にともなって、従来のミニコンの分野にまで、およんで来ています。メガ300はこのようなユーザーのために開発された高速ディスクシステムです。製品は、①インストール済みの330メガバイト磁気ディスク、②ケーブル、③SCSIインターフェースボード、④デバイスドライバ、⑤マニアルで構成されており簡単な操作で利用が可能となります。

※メガ600もあります。なお使用固定ディスクは日立DKシリーズです。

※MSDOSはマイクロソフト社の登録商標です。

特長と用途

■画像処理システムにおける画像ファイルの格納、処理に威力を発揮。
■パソコンCADシステムの本格的データベースとして利用。
■プリントサーバ、ファイルサーバの補助記憶として利用。
■パソコン会計処理システムの金銭データの格納ファイルとして。
■パソコンNCシステムのNCデータの格納、処理に利用。
■計測システムの日時データの記録、再生、処理に活用。

お問合わせは
■03-383-0456 山田まで
会社案内 ☎0120-383-234

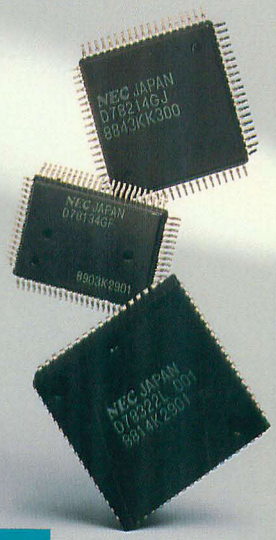
先人の遺産を未来に……
アイビックス株式会社

本　　社　〒164 東京都中野区本町5-17-13 石川ビル　TEL.(03)383-0234 FAX.(03)383-0239
情報技術センター　〒164 東京都中野区本町5-10-10　TEL.(03)383-0456 FAX.(03)383-0215
大 阪 支 店　〒532 大阪市淀川区西宮原1-6-20-208　TEL.(06)394-6631 FAX.(06)394-6630

資料請求No.3

C&C Computers and Communications

NEC



さまざまな民生機器や産業機器で使われる制御用マイコンに、いま、より高速、高機能化が求められています。「78Kシリーズ」は、好評の87シリーズの優れた機能、特長を受け継ぎ、さらに最近の市場ニーズに応えた新シリーズです。最新のアーキテクチャを駆使し、高速処理の実現や新しい処理形態を加えた高機能割り込みコントローラの内蔵。また各種命令も強化するとともに、ソフト資産の有効活用に貢献する、8ビットから16ビットへの移行が容易な78K命令セットの採用など、大きくパワーアップされています。8ビットと16ビットで構成される新シリーズは、多彩な品種展開で多彩なニーズに対応、開発サポートについても充実した開発環境を提供し、スピーディにシステム開発が行えます。お客さまの声にいちはやく応えるNECのシングルチップ・マイコンです。

高速化、多様化する制御分野に応える
78Kシリーズは、8ビットと16ビットの
新しい1チップ・マイコンシリーズです。

日本電気株式会社
お問い合わせは：半導体第一販売事業部、半導体第二販売事業部
半導体応用技術本部第一応用システム技術部
〒108 東京都港区芝五丁目29-11(日本電気住生ビル)☎(03)456-6111(大代表)
半導体応用技術本部第二応用システム技術部
〒540 大阪市中央区城見一丁目4-24(日本電気関西ビル)☎(06)945-3383(ダイヤルイン)
半導体応用技術本部専用マイクロコンピュータ技術部
〒210 川崎市幸区塚越3-484(川崎技術センター)☎(044)548-8888(ダイヤルイン)

N E C
シングルチップ・マイコン
78Kシリーズ

90周年。新たな未来へ……

資料請求No.4